MASTER IN COMPUTER SCIENCE

Complex Event Processing for Internet of Things
Open-Source Frameworks Analysis

Warszawski, Kenny

Award date:
2020

Awarding institution:
University of Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

Complex Event Processing for Internet of Things: Open-Source Frameworks Analysis

Kenny Warszawski

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Université de Namur
Faculté d'informatique
Année académique 2019–2020

Complex Event Processing for Internet of Things: Open-Source Frameworks Analysis

Kenny Warszawski

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Pierre-Yves Schobbens

Co-promoteur : Moussa Amrani

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Acknowledgements

I would like to thank in particular my promoters Mr. Amrani and Mr. Schobbens for their support in writing this master's thesis. They were able to answer my questions and get me back on track when I was lost. The meetings I had with Mr. Amrani and Mr.Schobbens allowed me to better understand the subject and find out how to approach it. Mr. Schobbens' theoretical course last year allowed me to understand the issues and the process to be followed during scientific research. I would also like to thank all those who were able to help in any way with the development of this thesis.

Résumé

Les systèmes informatiques sous-jacents à l'Internet des Objets impliquent de prévoir des mécanismes permettant de gérer une quantité massive d'informations provenant de différentes sources. Afin de gérer cela de manière efficace, le Complex Event Processing (CEP) vient s'intégrer dans ce genre de système afin d'optimiser la détection de situations complexes et d'en ressortir un évènement de plus haut niveau. Ainsi, une autre couche applicative pourra gérer cet évènement et appliquer le traitement adéquat.

Malheureusement, il est difficile de savoir quel framework utiliser pour une situation donnée. Dans ce travail, l'objectif est de déterminer quel framework permet de répondre à des besoins selon 3 axes: la prise en main, la maintenabilité et l'utilisation des ressources. Au travers d'une grille d'analyse permettant de couvrir ces 3 axes, des métriques vont être capturées depuis les frameworks Open-Source les plus populaires et toujours maintenu à ce jour. (Esper, Siddhi, Drools, Perseo)

A partir des données récoltées, une synthèse permettant de visualiser les forces et faiblesses de chacun des frameworks selon ces 3 axes sera fournie en sortie de ce travail.

Mots-clés: Complex Event Processing, CEP, Internet des Objets, IoT, Open-Source, Esper, Siddhi, Drools, Perseo

Abstract

The computer systems underlying the Internet of Things require mechanisms to manage a massive amount of information from different sources. In order to manage this effectively, Complex Event Processing is integrated into such systems to optimize the detection of complex situations and to bring out a higher level event. Thus, another application layer will be able to manage this event and apply the appropriate treatment.

Unfortunately, it is difficult to know which framework to use for a given situation. In this work, the objective is to determine which framework allows to meet needs in 3 axes: getting started, maintainability and resource usage. Through an analysis grid covering these 3 axes, metrics will be captured from the most popular Open-Source frameworks and still maintained to this day. (Esper, Siddhi, Drools, Perseo)

Based on the data collected, a synthesis allowing to visualize the strengths and weaknesses of each of the frameworks according to these 3 axes will be provided at the output of this work.

Keywords: Complex Event Processing, CEP, Internet of Things, IoT, Open-Source, Esper, Siddhi, Drools, Perseo

Contents

1 Introduction

Internet of Things is omnipresent and the amount of connected devices is increasing day by day. Internet of Things is different from a classic information system by its capability for integrating with the physical world. Those devices are used in many sectors (health, industry, transport, home automation, ...) and their users can be both professionals and individuals. Nowadays, we see the apparition of connected devices for home to facilitate our daily life. (e.g. Google Home, Nest, Philips Hue) Internet of Things is also an interesting technology for Smart Cities use case. For example, we can imagine a city where traffic lights are optimised with the city mobility to avoid traffic jams. However, an important problem arises in this type of information system. How is it possible to handle such an important data traffic efficiently ? Indeed, if an entire city has a huge amount of connected objects, the data flow to be processed is massive. Therefore, efficient data processing mechanisms need to be put in place to handle such flows.

The Complex Event Processing is part of the solution to handle such massive data flow. This technique makes it possible to determine complex patterns from several different data sources in record time. In the IoT, data is often extracted from many different sources, so events are not distributed in a single continuous stream. Therefore, Complex Event Processing is very efficient for this type of situation. In addition to being able to act on several sources, it can correlate events from several streams and when a pattern is detected, it is possible to send a more complex event that summarizes the current situation. To this day, there are a lot of libraries available to perform CEP operations, but it is difficult to determine which framework is best suited for a given context. The goal of this thesis is to help the reader to find the framework that best fits his need. Through analysis grids and implementation in selected libraries, this paper will try to answer this problem.

This thesis is organised as mentioned hereafter. The section 2 is dedicated at putting in place the background of the topic discussed in this article, namely Complex Event Processing for Internet of Things. All the fundamental concepts for a good understanding of this thesis will be reviewed. Then, in the 3 section, we will talk about the issues raised during the research. First, by posing the problem in the section 3.1, we'll get a general idea of the problem. Afterwards, we are going to address the research question that will try to answer the problem posed above. In this section, we will also discuss all the important aspects to consider when developing an IoT project and the scope of this thesis. Regarding the section 4, the methodology but also the tools to retrieve the information that will help to answer the problem question will be explained. Finally, the last parts will be dedicated to the explanation of the tests that will be carried out as well as their results and interpretation. Finally, this paper will end with a concluding note.

2 Background

2.1 Internet of Things

The Internet of Things(IoT) [13] is a novel paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications. The basic idea of this concept is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. – which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals.

The phrase "Internet of Things" was mentioned the first time by Kevin Ashton, the co-founder of the Auto-ID Center at MIT, as the title of one of his presentation. For him, today computers and the internet more generally are dependent from human intervention. Information technology

architectures are all designed around hardware, software interactions, etc. - but an important thing is generally set aside: people. People have a limited time, attention and accuracy. Hence, they are making mistakes about the real world. Thus, the data generated by humans is not the exact representation of the reality in an Information System.

A "Thing" can be any connected device from our daily life that is able to capture information from the physical world by sensors and send it through a network. Things are also able to receive tasks to execute thought different transport protocols. Information sent by sensors can be kept by another layer able to process data and execute other complex tasks.

2.1.1 Architecture

The architecture of an Internet of Things solution can be implemented with various technologies and in very different infrastructures. However, a generic high level architecture is common to all IoT projects as illustrated on figure 1.

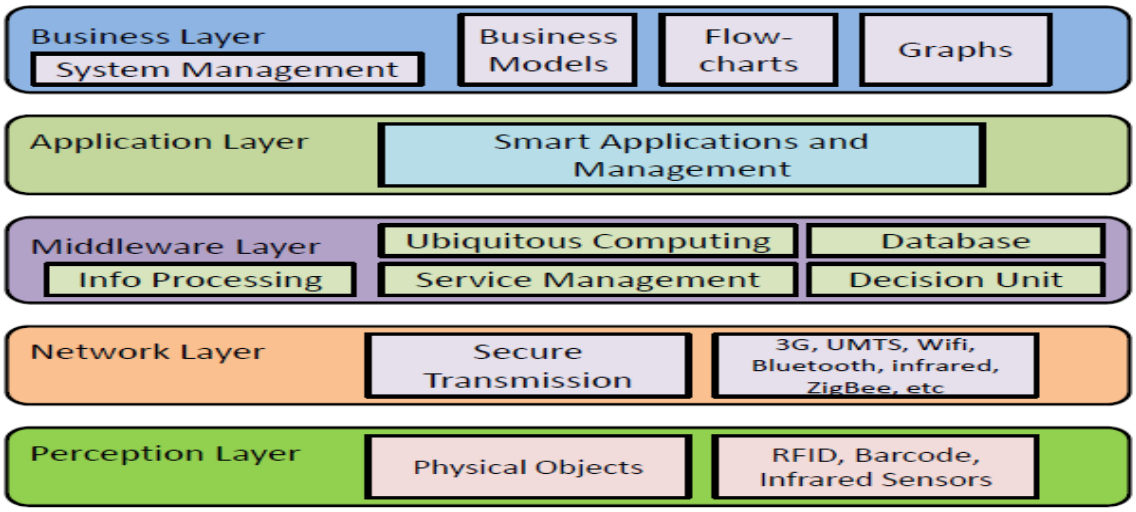

Figure 1: Generic Architecture. (Source: [16])

Perception Layer: The perception layer or device layer is the layer where physical objects will be able to capture data via sensors. These sensors are able to collect real world data. These data can be or not be processed upstream to add meta-data by the connected object itself and will be forwarded to the Network Layer.

Network Layer: The Network Layer or Transmission Layer. This layer transfers the information from to Perception Layer to the Middleware Layer. This transfer can be accomplished by different communication technologies like Wi-Fi, 3g, Bluetooth, etc.

Middleware Layer: The Middleware Layer is the common layer for all physical objects of an IoT architecture. The middleware is responsible to take automatic decisions based on information coming for the devices and store the results in a database.

Application and Business Layer: The Application Layer is where smart applications are running to produce complex processes. These applications will consume data from connected objects previously processed by the middleware to act effectively on a specific situation. These application can transmit actions to execute on a particular type of service. The Business Layer is important for the global management of the system. This one is rather a layer of global monitoring which produces graphs and statistics at the business level.

2.1.2 Data Flow

In order to see more clearly in this generic architecture, it is possible to zoom in on the data flow from a device to a system. The figure 2 illustrates data capture via sensors. These data will be transited to a Computation and Processing System through a communication layer.

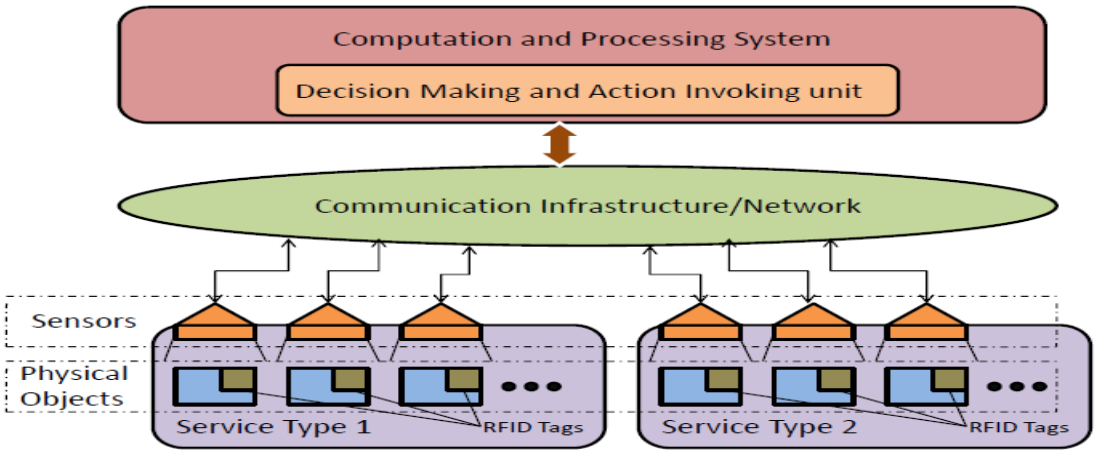

Figure 2: Simple IoT Architecture. (Source: [16])

This thesis is focusing on the "Computation and Processing System" layer because at that stage, the data processing is the most critical and the data flow is the highest. Obviously, the Communication Layer must be robust enough to transmit the data efficiently but this topic will not be tackled in this thesis. The data flow through an IoT system can be very high because, as mentioned above, the number of connected objects is increasing. It is therefore necessary to have an efficient method to handle a massive amount of events on several streams. It is in this processing layer that Complex Event Processing will play an important role in the IoT architecture.

2.2 Complex Event Processing

Complex Event Processing (CEP) [29] is a set of methods and techniques for tracking and analysing real-time streams of information and detecting patterns or correlations of unrelated data (complex events) that are of interest to a particular business. The complex event processing is the engine of real-time applications that needs to have real-time information from an environment to respond as fast as possible. A standard Request/Reply with synchronous processes cannot correlate different type of events and respond in a timely manner. Complex Event Processing mechanisms are often used in Event Driven architectures. This architecture fits with the Internet of Things needs. This kind of architecture is driven by events sent by connected objects. Then, the services at the application layer are consuming those events.

2.2.1 Events

The events coming directly from the sensors are call raw events. The raw events are the events coming from the real world. It may represent a simple real-world event or a complex real-world event. Those raw events often need to be pre-processed before integrating a more sophisticated system. Without the pre-processing it would be difficult for a system to understand the information correctly and correlate it with other events. A strict semantic between different type of events have to be put in place for the whole IoT system. That is why a complete ontology should

structure all the domain application. Generally, two types of events can occur in the system: Internal and External. The internal events are the events sent by the application layer. The external event are these sent by the "physical" sensors.

2.2.2 CEP Engine

Events are the trigger of one or multiple process in an IoT solution. The CEP engine is consuming those events sent by the physical objects on the Communication Layer. This engine defines a set of rules where each received event will be evaluated by the appropriate rules for that event. A rule represents a complex condition that can be based on multiple events and on a time window. Basically, a complex event processing engine correlates several types of events over time. Then, if the pattern defined in the rule is matching with the ingested events, a complex event is sent. A complex event [22] is an event that summarizes, represents, or denotes a set of other events. The complex events generated can themselves triggers another CEP rule or an application process. For example, if a temperature sensor captures that the average temperature of the last 60 seconds is higher than 45 degrees, an event saying that fire is detected will be produced. Then an alarm can be triggered to warn people in the house and a notification can be sent to the fire department.

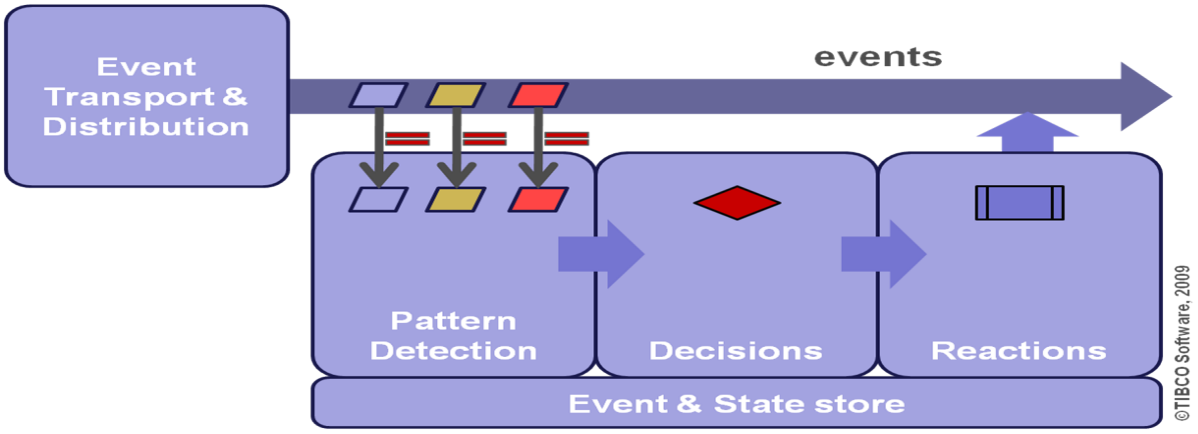

Figure 3: CEP Pattern Decision Reaction. (Source: [33])

Technologies implementing CEP concepts are often querying Time Series Databases where each event received is stored. Those databases are optimised for queries based on a time window.

2.2.3 Event Pattern Languages

An Event Pattern Language (EPL) is a language used by CEP engines in order to describe the relations between events matching a specific pattern. The syntax of EPL's are quite similar to SQL queries. Example 1 shows a composite event called Thermometer, which carries two pieces of information: the value of the sensed temperature as a real and the date it was sampled. The query below selects all dates of events where the temperature was sensed above 30°C.

Example 1:

```
Thermometer (temperature: real; date: Date)

SELECT Thermometer.date
FROM DataSource
WHERE temperature > 30
```

A plenty of primitive operators exists in Event Pattern Languages to describe the rule's behaviour. The following operators are coming from ThingML framework but are available in most of CEP frameworks:

Selection: [18] Filters relevant events based on the values of their attributes. For example, we can select all Air Pressure events between 101300 Pa and 101400 Pa.
Projection: [18] Extracts or transforms a subset of attributes of the events.
Window: [18] Defines which portions of the input events to be considered for detecting pattern. For example, a rule can concern the last 30 seconds events.
Conjunction: [18] Consider the occurrences of two or more events. This operator is basically a logical 'AND' operator.
Disjunction: [18] Consider the occurrences of either one ore more events in a predefined set. This operator is basically a logical 'OR' operator.
Sequence: [18] Introduces ordering relations among events of a pattern which is satisfied when all the events have been detected in the specified order.
Repetition: [18] Considers a number of occurrences of a particular event.
Aggregation: [18] Introduces constraints involving some aggregated attribute values. For example, the average temperature of all Weather events is an aggregation.
Negation: [18] Prescribes the absence of certain events. This operator is basically a logical 'NOT' operator.

2.3 Computing Levels

To cope with low latency challenges required by IoT systems, CEP engines may be strategically deployed at various levels of the IoT architecture stack, depending on the constraints and bandwidth available: at the level of the cloud, the fog, or on the edge of the system.

2.3.1 Cloud Computing

Cloud Computing places the computing complexity at the cloud level. All the data coming from the sensors have to transit by the connected object, sometimes by a middleware and then it can be consumed by a CEP Engine in the cloud. This work-flow is illustrated clearly by the figure 4. The Cloud Computing is common for near real-time applications where a little bit of latency is not critical.

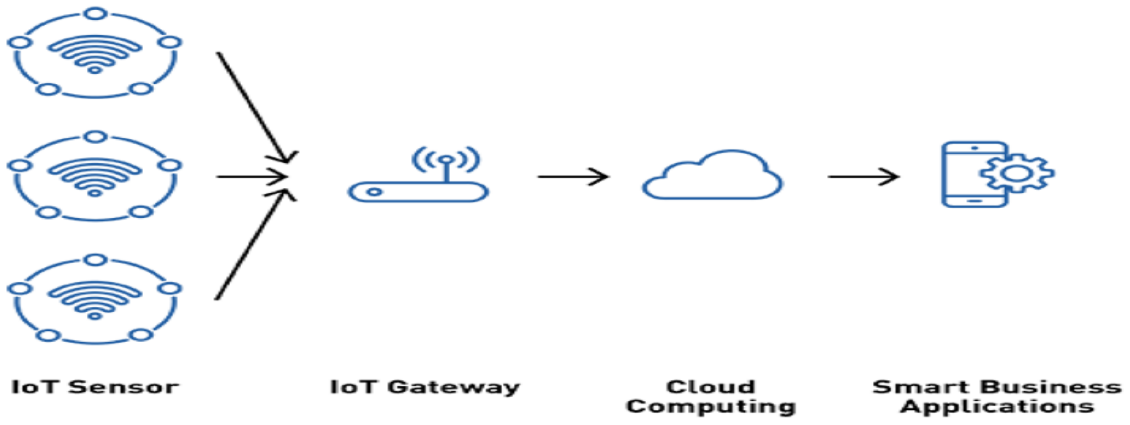

Figure 4: Cloud Computing. (Source: [3])

This solution is optimal for the near real-time cloud monitoring because all events coming from the devices are sent to the cloud. Then a precise metrics history can be retrieved. Obviously all data persisted have a cost. This type of solution is extremely demanding in storage space if all the history is persisted.

Naturally, Cloud Computing can be coupled with other levels of computing and it is often the case. The Cloud Computing can be placed on the top of a multi site IoT solutions where millions of data are coming from thousands of connected objects in the world. Then, this solution can reflect a High Level overview of a global IoT system and available all over the world.

2.3.2 Fog Computing

Fog Computing places the computing intelligence at the local network level. Typically, the complexity is put between Cloud and IoT sensors. The devices are sending their data to a Middleware and the CEP engine is processing the data locally and send the processed events to a service running in the Cloud or on a local server. This method offers the CEP performances on a closer level where the data are created. Then, for applications where latency is crucial, Fog Computing can fit with their business needs. For example, auto mobile or aerospace sectors are more and more connected and the reactivity of their system needs to be as fast as possible. Compared to a device-to-cloud architecture, placing processing closer to the devices can reduce the latency since the physical distance is shorter and potential response time in a data center can be removed. Compared to a device-only architecture, latency can be reduced since computation intensive tasks that take a long time on resource-constrained sensor devices can be moved to more capable fog computing nodes. The figure 5 illustrates this architecture. [9] The motivation can also be to keep the latency predictable.

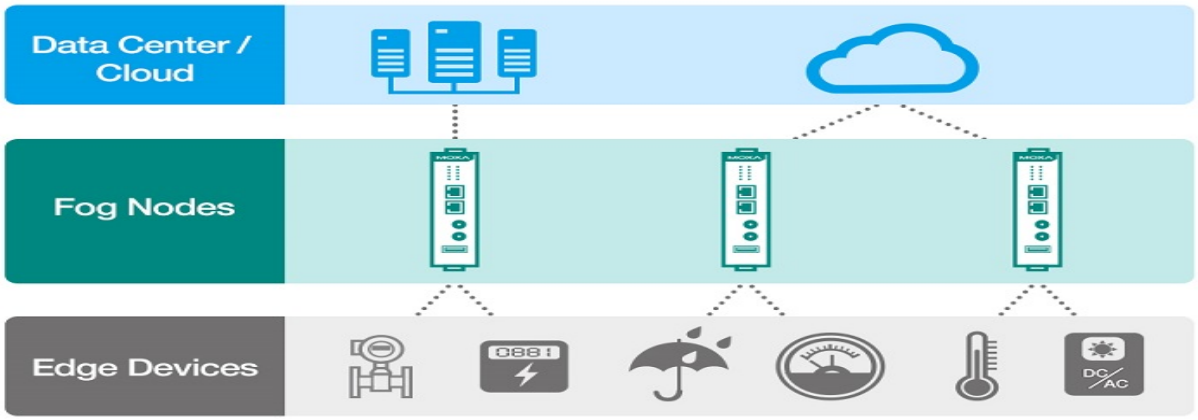

Figure 5: Fog Computing. (Source: [14])

It can happen that the edge devices are sending too much data at very high rate. Therefore the processing engine at fog level is not treating the data fast enough because the processing of a huge amount of data is taking too much time. Complex Event Processing can be put at the Edge devices level. The terms Fog and Edge computing have different meanings depending on the article. In some articles, edge computing refers to the technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services.[30] The limit between Fog and Edge computing is not clear. In this state-of-the-art we consider the Fog to be a higher layer on the top of physical devices like a server on a private network.

2.3.3 Edge Computing

Edge Computing places the computing complexity at the Physical Layer. It can be an IoT gateway where data is pre-processed before being pushed to the Cloud or whatever. For example, in e-health some sensors can be put in a human body and the information coming from the sensors are sent to a smartphone. The smartphone is processing those information before sending them. This method reduces the network traffic to the Fog Nodes and/or the Cloud because less information will be transferred. Therefore it reduces the risk of data congestion and the CEP engines are not flooded. Edge Computing is generally used by industries in Cyber Physical Systems[1].

2.4 Semantic Publish-Subscribe Architecture

The article "A Semantic Publish-Subscribe Architecture" [28] suggests an interesting asynchronous architecture based on the Publish-Subscribe design pattern. The publish-subscribe pattern in software architecture is a messaging pattern where the sender of the message does not know the receiver. On the other side, the receiver does not know the sender as well. The publisher is sending a message on a message queue where subscriber(s) are listening to. In Internet of Things solutions, the producers and consumers can be the connected objects themselves but can also be an application layer in the Cloud for example. The connected objects are publishing events on event topics and the application layer processes those events.

2.4.1 Overview

The SPS architecture prone the modularity, extensibility and cost-effective vision. It splits the physical world from its digital representation. The primary characteristic consists of splitting clients into 3 categories: consumers, producers and aggregators. The business logic is driven by the aggregators. The producers and the consumers are the bridge between the physical world and the virtual representation of the systems. This principle keeps the business logic at the aggregator level in order to keep the clients and producers as simple as possible. It gives to the producers the unique responsibility to send data to an upper layer without executing any business logic. Hence, this system is easier to extend because only the aggregator layer is impacted if a business feature is added. Then, the consumers and producers can be shared by different applications. All these notions are represented on figure 6.

[1] Cyber-Physical Systems (CPS) are integrations of computation and physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [19]

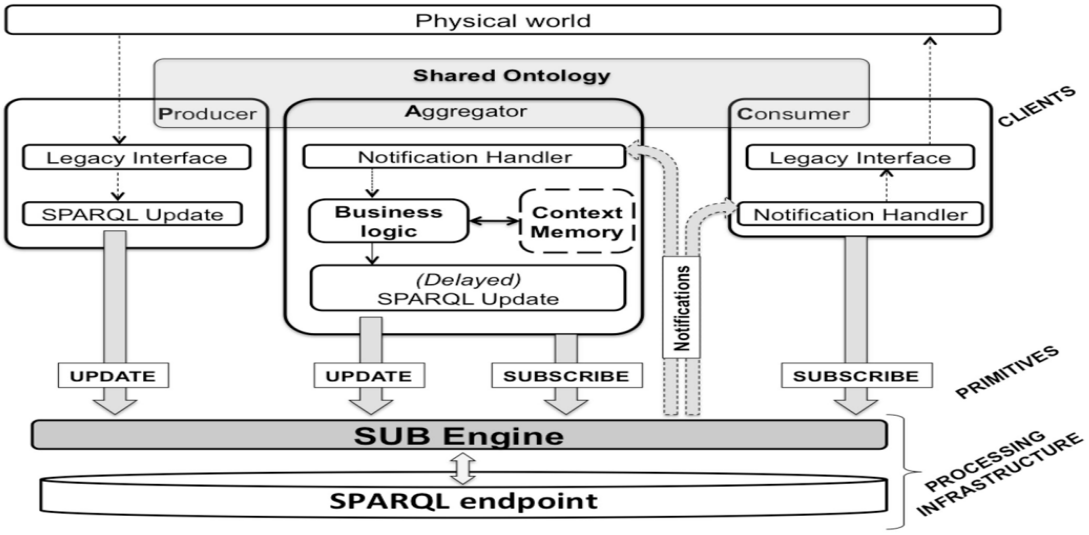

Figure 6: Semantic Publish Subscribe Architecture. (Source: [28])

The role of the producers is to collect data from sensors. The producer can optionally make a local processing where it encapsulate the data to a semantic format. After that, the producer send the information to the SUB Engine that will store the data in a Triplestore. Contrarily, the consumers are listening to the concrete events coming in the SUB Engine that can be a result of a processing done by an aggregator. After receiving an notification, the consumers extract the raw data and send it to a devices through a legacy interface.

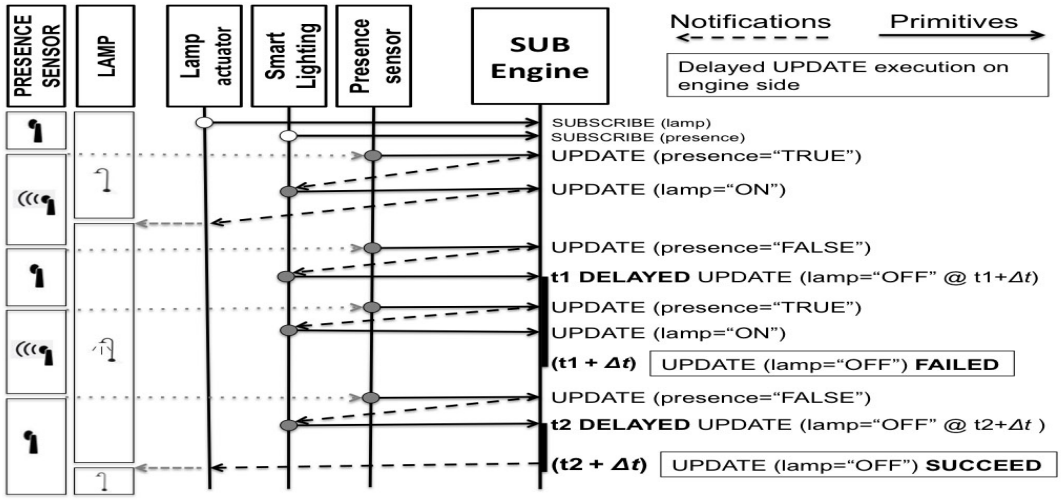

Figure 7: Sequence diagram of Smart Lightning. (Source: [28])

The aggregators are listening to the events sent by the producer through the SUB Engine. Each aggregator subscribes to each event type they need for their business process. The fig.7 exposes a simple use case of Smart Lightning where each type of clients described previously is used. The presence sensor acts as a producer and store the presence state in the RDF Store. The Lamp actuator acts as a consumer and subscribe to the Lamp events. The Smart Lightning acts as an aggregator to handle incoming events and triggers a business process based on those events.

Scenario:

1. The Lamp Actuator is subscribing to the Lamp events and the Smart Lightning aggregator is subscribing to the Presence events.
2. The Presence sensor is Sending an Event to the SUB Engine that a presence is detected.
3. The presence event is captured by the Smart Lighting aggregator that will send an event to switch on the lamp.
4. The "Switch ON the lamp" event is captured by the Lamp actuator that communicate the information to the connected lamp.
5. The presence sensors is sending that no presence is detected.
6. The Smart Lightning schedules to send an event to switch off the light after x seconds if no presence is detected during that time laps.
7. A presence is detected, then the Smart Lightning cancels his event scheduling.
8. The presence sensors is sending that no presence is detected.
9. The Smart Lightning schedules to send an event to switch off the light after x seconds if no presence is detected during that time laps.
10. The Smart Lightning Aggregator sends the "Switch off lamp" event.
11. The Lamp actuator is switching off the light.

2.5 Domain of applications

2.5.1 Smart Building

The article [1] proposes an Internet of Things architecture to reduce building's energy consumption. Studies demonstrated that using ICT technologies could reduce the energy consumption of a building by 70 percent. Different automatic processes have been put in place to control the ventilation, heating and air conditioning control based on data coming from the sensors. Different algorithms were tested to provide a better way to maintain the heating and air quality of the building with the lowest energy consumption.

The article [5] is focusing on a distributed Complex Event Processing Architecture. The work is based on a smart Building use-case. The paper highlights the fact that it is not appropriate to use a centralized architecture for an IoT environment. Having a centralised server means having devices sending an enormous quantity of raw data to one unique server. Hence, the server must have enough powerful hardware resources. The ability to scale such systems efficiently is not possible. The paper proposes to distribute the processing load between multiple servers. Some event processing is performed upstream in order to filter only the payload data to other servers that have another processing responsibility. The proposed solution allows users to define the rules and actions to be executed from a graphical user interface.

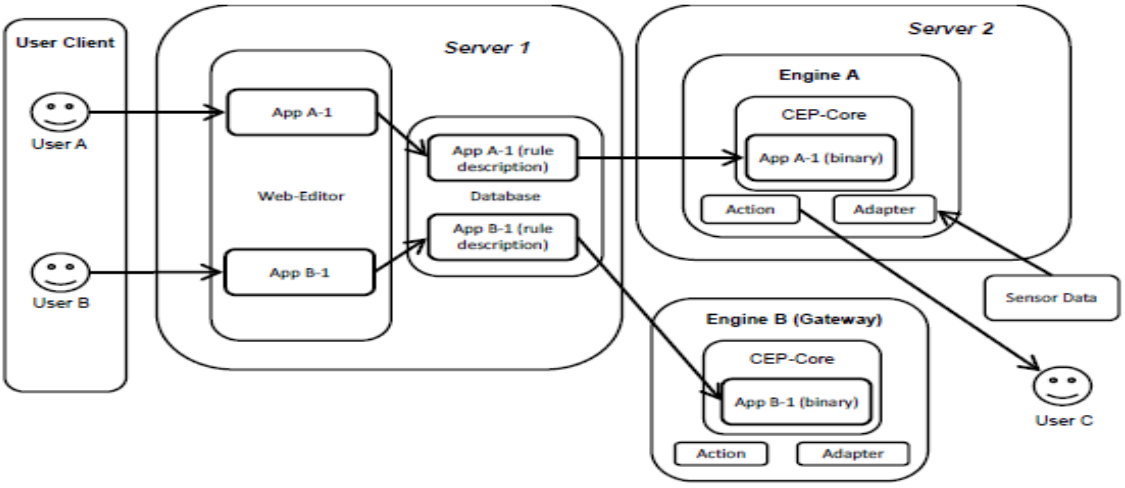

Figure 8: Distributed Complex Event Processing. (Source: [5])

2.5.2 Healthcare

Healthcare is a sector where the investment for the Internet of Things is significant. More and more hospitals are offering connected devices to monitor the health status of their patients. Besides, the article [17] presents the promises of Fog and Edge Computing in the field of health. Fog Computing brings the possibility to handle dynamic contexts. Medical devices main responsibility is to collect data from patients. Therefore, all data processing inside devices should be minimal to provide the minimum usage of the battery. Fog computing provides flexibility in data processing. If an update needs to be made to the data processing level, the only impact is located on the Fog Level.

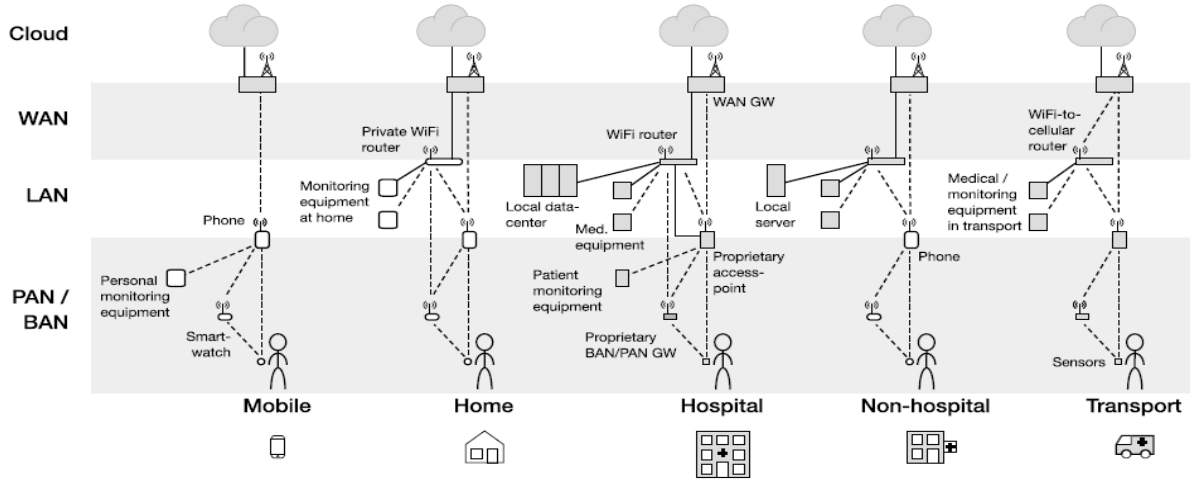

Figure 9: Healthcare Fog-Computing. (Source: [17])

Information from sensors inside the patient must be available in different places. The patient must be able to be notified directly at home if a potential problem is detected. Therefore, connected medical objects must be able to connect to different devices: smart phone, home, hospital, etc. (see: figure 9)

2.6 Research Methodology

Main resources :

- **IEEE Xplore**
- **Springer**

Step 1 - Selection
The methodology used for this research was inspired by the Systematic Mapping Study. [26] Initially, the first query done on the IEEE Xplore Digital Library was : Complex Event Processing AND Internet of Things. This result gave too much results, around 200. To reduce this number, only the papers published between 2014 and 2019 were chosen. The Internet of Things is a subject that is continuously evolving, then it is more interesting to focus on the newest technologies. This filter excluded around 40 articles. After that the 100 most cited were chosen for the second step.

Step 2 - Pre-Reading
The second step consisted of reading all the 100 abstracts of the articles. Then, the most relevant for the subject were selected. Only fourteen articles were selected out of the hundred.

Step 3 - Careful reading
After that, a first reading of all articles revealed unnecessary articles and only few of them (6) were deeply analysed and summarized. After summarizing and a better understanding of the problematics for the subject, it was necessary to find a glossary [22] with all technical definitions. A lot of authors are employing various words for the same concept and it was sometimes fastidious to correlate all the information together to have a good understanding of all information.

The second search on IEEE Xplore consisted of searching other Data processing techniques in the domain of Internet of Things. The query was : ((((Internet of Things) AND Data Processing) NOT Complex Event Processing)). With the filters : 2014 to 2019 and the 100 most cited papers. For this search, the same methodology was applied.

After all this research, the writing of this state-of-the-art started. Sometimes, some information were missing to correlate some parts with others or some concepts weren't explained clearly. Then, some small additional researches were done during the writing. Those research were either a search in a scientific library or from a reference in an analysed article.

3 Research

3.1 Problematisation

Through the state-of-the-art, a problematic aspect in the Complex Event Processing for Internet of Things domain has emerged. Due to the diversity of CEP frameworks on the market, it can be quite difficult to know which one is used for a given problem. Since this choice depends on many aspects, it is not easy to know if the chosen framework will be adapted for a given situation.

3.1.1 Aspects

Getting Started
When searching for a framework, it is crucial to know whether or not it is capable of performing certain CEP operations. It is also important to know if these operations can easily be used with each other in order to detect more complex patterns. Those information can be found out through the documentation and in examples that can be found on the framework website. Other information such as compatible languages or access to support must be easily retrieved. It is therefore important that all this information is available.

Code Maintainability
When developing a project, it is important to determine whether the framework used can facilitate the maintainability of the code. Of course, this factor also depends on the skills of the developer, but a prior logical breakdown can facilitate this work. The presence of examples, templates and guidelines is a plus to facilitate the maintainability of the code. The framework must also have a sufficiently clear and precise documentation so as not to mislead the developer in his design choices.

Resource Usage
Resource usage is also an important aspect when choosing a framework. When a solution must run on a given machine, the framework must match the specifications of the target machine. It is inconceivable to invest in a framework that consumes too much CPU or RAM for example. It is also necessary to check its use in the disk space and the GPU usage.

3.2 Question

The purpose of this master's thesis is to answer the research question:
Which Complex Event Processing framework should be used for an IoT project ?

To answer this question, it is possible to sub-divide this question into 3 sub-questions corresponding to the aspects of the problematisation (3.1) :

Which framework allows to develop an IoT solution in the easiest way ?
Which framework allows to develop the most maintainable IoT solution ?
Which framework allows to develop an IoT solution that uses the least amount of resources ?

Through these questions, different trends can be identified between the CEP frameworks. Thus, depending on the criteria expected for the IoT project, the decision to choose a framework can be motivated by these 3 axes.

3.3 Scope

The scope of this research will still be limited and will not compare all CEP frameworks on the market. This research will be limited to the most popular Open-Source frameworks. Of course, only those frameworks that are still maintained today will be analyzed. Proprietary frameworks as well as non-maintained frameworks will be excluded from this research.

The chosen frameworks will be mostly "detection-oriented" and not "computation-oriented" ones. In fact, these two characteristics are inherent to the difference between a CEP framework and an

ESP framework. By definition, an Event Stream Processing framework [21] will focus on the computation of events coming from an event stream. An event stream is a sequence of events ordered by time, such as a stock market feed. A CEP framework by definition will focus on computing in an event cloud. An event cloud is the result of many event generating activities going on at different places in an IT system. A cloud might contain many streams. A stream is a special case of a cloud.

Unfortunately, today the difference between the two worlds becomes difficult to distinguish. In a complex system, the two types of technologies have to cohabit together. In addition, ESP platforms [21] are increasingly tending to include specific operations from Complex Event Processing in order to provide a complete solution. In the literature, these terms are not clearly defined. For the purposes of this submission, it is therefore considered that ESP describes the first layer of event processing. We are talking about high-volume, high-velocity event streams. ESP solutions allow to apply continuous querying on event flows as close as possible to their appearance to perform simple operations (normalization, filtering ...) and in real time (without latency) to trigger responses, microprocesses and immediate actions when the desired pattern is detected. Complex Event Processing [6] generally intervenes downstream of the ESP, in the 2nd layer and constitutes the core processing of event processing, where the complexity is concentrated. Where the ESP is "confined" to mass processing of events of the same nature and without complex correlation, the CEP is able to correlate complex events of a very varied nature. The CEP thus enables the detection of business situations based on complex patterns, requiring asynchronous correlations of multi-source event flows.

4 Methodology and Tools

In order to be able to answer the various questions raised by the research, it is necessary to apply a rigorous testing methodology. Firstly, a research and selection phase of the Open-Source frameworks of Complex Event Processing was carried out. Then, in order to compare these selected frameworks in an equivalent way, CEP patterns will be implemented in these different technologies. These patterns will be examples to show the complexity of the implementation of the operators mentioned in the state of the art phase (section 2.2.3). In the section 5.3, these scenarios will be explained in a more precise way. From these implementations and the metrics that will be explained in the section 5.2, it will be possible to factualize the differences as well as the strengths and weaknesses of each implementation. In the same section, we will see that technical metrics will also be captured to determine the physical resource consumption of each framework. To do this, different test scenarios will be performed to simulate a more or less heavy workload.

In order to find the Open-Source CEP frameworks, the GitHub website was used. In this selection, the keyword "Complex Event Processing" was used. In order to sort in this list, only the frameworks still maintained today are kept in this sort. To do so, repositories still having commits in 2020 are considered to be maintained. At this stage, a list of 34 repositories are listed by GitHub.(`https://github.com/search?l=&p=1&q=Complex+Event+Processing++pushed%3A>2020-01-01&ref=advsearch&type=Repositories`) In this list, repositories that are not Complex Event Processing libraries must be removed. In these repositories, it is also necessary to sort among the frameworks with documentation. Libraries that do not have documentation are considered as not having the necessary maturity to be analyzed. In the next section, the frameworks corresponding to these search criteria will be analyzed.

In addition to these frameworks, popular open source framework for Event Stream Processing exposing Complex Event Processing operations will be added for analysis.

5 Evaluation

5.1 Open-Source Frameworks

The objective of this section is to analyze the open-source frameworks found during the research and to analyze their relevance for the context of this master's thesis.

5.1.1 Esper

Esper [11] is a language, compiler and runtime for complex event processing (CEP) and streaming analytics, available for Java as well as for .NET. The design priorities for Esper are:

- Low latency and high throughput.
- Expressiveness, conciseness, extensibility of the EPL language.
- Compliance to standards and best practices.
- Light-weight in terms of memory, CPU and IO usage.

As part of this research, the implementation in Java will be carried out. The EPL language used by Esper being similar in Java and .NET, this choice was made by simple affinity with Java.

GitHub link: `https://github.com/espertechinc/esper`

5.1.2 Siddhi

Siddhi [31] is a cloud native Streaming and Complex Event Processing engine that understands Streaming SQL queries in order to capture events from diverse data sources, process them, detect complex conditions, and publish output to various endpoints in real time. Siddhi can run as an embedded Java and Python library, as a micro service on bare metal, VM, and Docker and natively in Kubernetes. Siddhi provides web-based graphical and textual tooling for development.

For the purpose of this research, the implementation as a Java library will be performed. In addition, the realized definitions will also be tested via micro-service export in a containerized environment.

GitHub link: `https://github.com/siddhi-io/siddhi`

5.1.3 WSO2 CEP

WSO2 CEP [35] is a lightweight, easy-to-use, open source Complex Event Processing server. It identifies the most meaningful events within the event cloud, analyzes their impact, and acts on them in real-time. It's built to be extremely high performing with WSO2 Siddhi and massively scalable using Apache Storm.

During the analysis, it was decided not to analyze this solution because the engine of Complex Event Processing is Siddhi. Moreover, WSO2 is the main maintainer of Siddhi. It is therefore not interesting to perform additional tests for this framework. These would be redundant with what is done in the Siddhi tests.

GitHub link: `https://github.com/wso2-attic/product-cep`

5.1.4 Drools

Drools [10] is a business rule management system with a forward-chaining and backward-chaining inference based rules engine, allowing fast and reliable evaluation of business rules and complex event processing. A rule engine is also a fundamental building block to create an expert system which, in artificial intelligence, is a computer system that emulates the decision-making ability of a human expert.

Drools exposes its functionalities via Java libraries. For the implementation of this framework, a maven project will be defined to apply the different CEP patterns.

GitHub link: `https://github.com/kiegroup/drools`

5.1.5 Apache Flink

Apache Flink [34] is a framework for stateful computations over unbounded and bounded data streams. Flink provides multiple APIs at different levels of abstraction and offers dedicated libraries for common use cases. This framework is very popular and is originally an event stream processing framework. By the way, it is offering a complex event processing library for pattern detection called FlinkCep. FlinkCEP is the Complex Event Processing (CEP) library implemented on top of Flink. It allows you to detect event patterns in an endless stream of events, giving you the opportunity to get hold of what's important in your data. [12]

Apache Flink is a well-known Event Stream Processing framework on the market. It includes Complex Event Processing libraries for processing a stream. Unfortunately, a POC concluded the fact that Flink is not suitable by itself for determining a complex rule based on multiple streams. In fact, Flink cannot act as an engine for Complex Event Processing. It remains at the Event Stream Processing layer. Some Architectures based on Kafka and Flink enable the ability for Flink doing Complex Event Processing but it still remains doing operation on one single stream and not from multiple sources at a time.

GitHub link: `https://github.com/apache/flink`

5.1.6 Perseo

Perseo [25] is an Esper-based Complex Event Processing (CEP) software designed to be fully NGSIv2-compliant. It uses NGSIv2 as the communication protocol for events, and thus, Perseo is able to seamless and jointly work with context brokers such as Orion Context Broker.

Perseo is based on Esper as engine for Complex Event Processing and exposes an HTTP REST Api to add patterns at runtime. It is defined as a CEP platform for Internet of Things. It is therefore interesting to test this framework. There is no implementation in a specific programming language. The implementation will consist of configuration via json payloads for the definition of events and CEP patterns.

GitHub link: `https://github.com/telefonicaid/perseo-fe`

5.1.7 Hurence: Logisland

Logisland [20] is a scalable stream processing platform for advanced realtime analytics on top of Kafka and Spark. LogIsland also supports MQTT and Kafka Streams (Flink being in the

roadmap). The platform does complex event processing and is suitable for time series analysis. A large set of valuable ready to use processors, data sources and sinks are available.

According to the documentation and the various information found about this framework, Logisland is not a framework for Complex Event Processing. This solution is mainly oriented Event Stream Processing but does not allow the detection of complex patterns and to return a higher level event based on several sources. This framework will not be analyzed.

GitHub link: `https://github.com/Hurence/logisland`

5.1.8 Clover-Group: TSP

TSP [32] is a Time Series Patterns search engine. This engine uses Complex Event Processing techniques for searching pattern matches from time series database.

This framework could be interesting to analyze in the context of Complex Event Processing for Internet of Things but it does not offer a complete documentation. This one is still in progress. TSP is still in beta version (v0.x.x). It means the project has not enough maturity for analysis purpose. This framework will not be analyzed in this thesis.

GitHub link: `https://github.com/Clover-Group/tsp`

5.2 Criteria and Metrics

In order to be able to quantify in a factual way the complexity of the implementation of the frameworks, it was decided to subdivide them into several categories. A general category will highlight some more general metrics about the framework and its ecosystem. Then, code metrics will be used to judge the quality of the code produced. Finally, technical metrics about the resources used will be described.

5.2.1 General Metrics

In order to be able to determine if a framework is easily usable by a lambda developer, in this section, some general information about frameworks will be captured. To remain as agnostic as possible of the developer's profile, it has been chosen to divide these criteria into 3 main axes that will be measured by objective metrics.

Documentation

Regarding documentation, we will try to set a scale to be able to compare frameworks between them. For this point, we will therefore set different criteria:

- **Architecture:** Is the architecture and/or the functioning of the framework explained ?
- **Quick start:** Is there a quick start guide available ?
- **CEP Operations:** Is there an explanation of the different CEP operators available in the framework? If so, are there code examples ?
- **Concrete use-cases:** Does the documentation provide concrete use-cases? If so, are there code examples ?

Support

In the context of a development, it may happen that a problem is encountered. This is why it is important that a framework is provided with some kind of support that allows to unlock the developer from this kind of situation.

- **Personal support:** Is there a customized support solution ? If so, by which channels ?
- **Community and Forums:** Are there channels to communicate with the community? If so, through which channels ?

Portability

As mentioned earlier in the report, it is important to determine if the chosen framework allows to be easily deployed on a target machine. In this case, we will determine the portability of the framework using the following criteria:

- **Programming Languages:** In which programming languages is it possible to integrate this framework ?
- **Operating system:** Under which operating system is it possible to run the code ?

5.2.2 Code Metrics

Regarding code metrics for characterizing code maintainability, it is very difficult to quantify precise measurements. Many CEP frameworks offer their service in the form of a DSL that is similar to an SQL query for streams. These DSLs often make it possible to offer some abstraction on the real complexity of the operations carried out in the background. In this domain, it is therefore difficult to capture cyclomatic complexity, maintainability index, inheritance depth or coupling between classes. In addition, some frameworks offer a platform to execute these queries and extract an executable from them. These metrics will therefore not be taken into account because these criteria cannot be compared between all the frameworks. However, another metric can be taken into account:

- **Lines of code:** How many lines of code are needed to write a pattern ?

5.2.3 Technical Metrics

The last axis concerns purely technical metrics. Measurements of physical resource usage are very important in order to be able to determine the minimum machine to support a more or less important load. For this purpose, different test scenarios will be set up to highlight these consumptions:

- **10 events per second during 2 minutes**
- **100 events per second during 2 minutes**
- **1000 events per second during 2 minutes**
- **10000 events per second during 2 minutes**
- **100000 events per second during 2 minutes**

For each scenario, the average resource consumption for each following metrics will be measured:

- **RAM Usage:** What is the use of Random Access Memory ?

- **Disk Usage:** What is the use of Disk ?
- **CPU Usage:** What is the use of Central Processing Unit ?
- **GPU Usage:** What is the use of Graphics Processing Unit ?

From these statistics, graphs for each CEP pattern can be produced.

5.3 CEP Patterns

To remain agnostic of particular syntaxes of EPLs, we specify the patterns we use to evaluate CEP/ESP frameworks using UML's Interaction Diagrams, where the actors are played by the sensors communicating with the CEP/ESP engine, and the interactions represent events transiting in the IoT system. Each pattern is designed to capture one particular EPL operator, or interesting combinations.

5.3.1 Water Level Overflow (Selection)

Figure 10 depicts a selection pattern for detecting a river overflow, using a water level sensor. The sensor periodically sends an event with the current value; when this value exceeds 20, the system issues an overflow notification event including the river's name. This pattern is inspired by the article [7].

Actors: System, Water level sensor
Operations: Selection

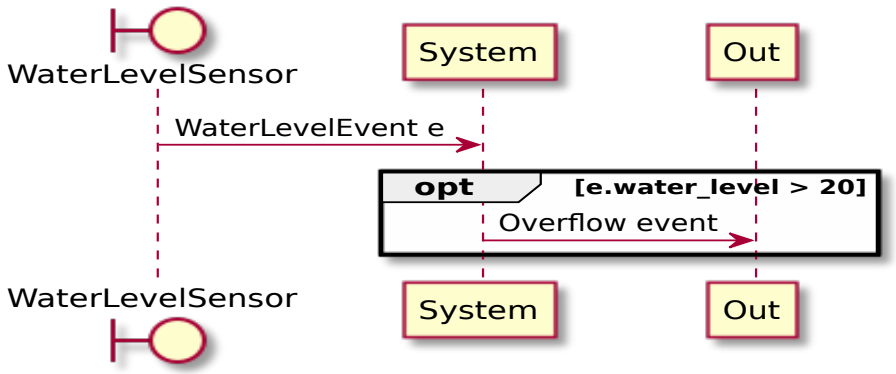

Figure 10: Water Level Overflow sequence diagram.

5.3.2 Smart Lightning (Selection, Window, Negation)

Figure 11 depicts the following scenario: When a motion sensor detects a presence, it signals it to the system. Thus, the system turns on the light. Subsequently, if in a time window (30 seconds) the motion detector no longer detects a presence, the system will switch the light off. If a presence is detected again during this window, the lamp remains on. This scenario is inspired by the article [28].

Actors: System, Motion sensor, Lamp
Operations: Selection, Window, Negation

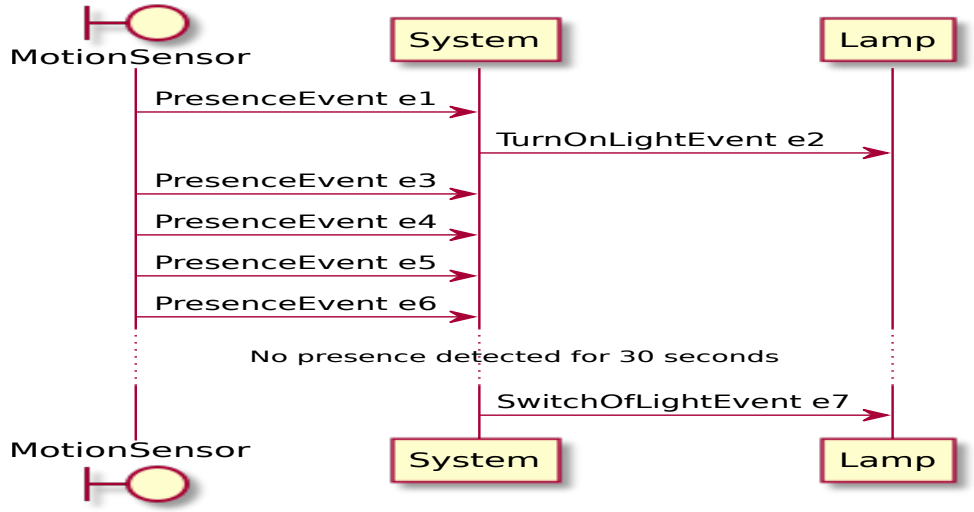

Figure 11: Smart Lightning sequence diagram.

5.3.3 Fire Detection I (Selection, Window, Aggregation)

Figure 12 illustrates a fire detection pattern. Temperature data is collected by a sensor. This data is sent to the system where an average of this data is calculated for the last 10 minutes. If this average value is higher than a threshold value (40), an event is sent to the alarm. This pattern is inspired by the article [23].

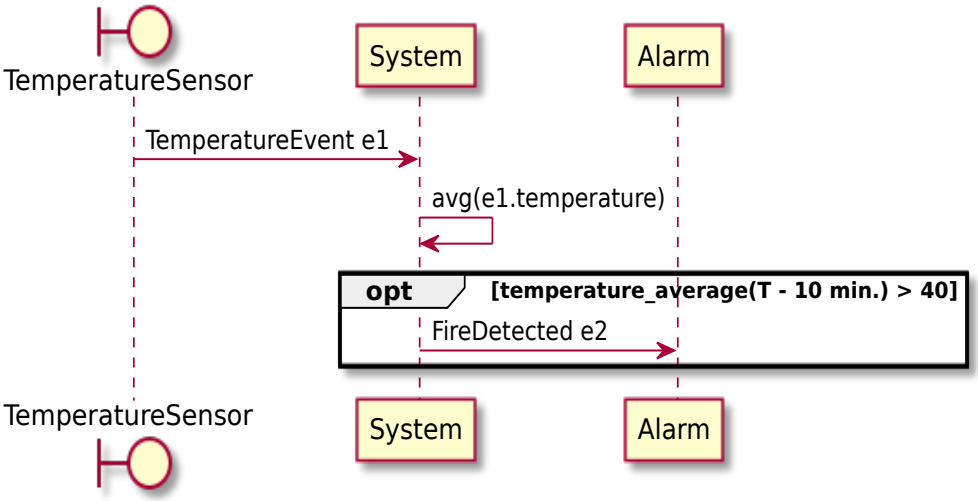

Figure 12: Smart Fire Detection I sequence diagram.

5.3.4 Fire Detection II (Selection, Window, Aggregation, Conjunction)

Figure 13 depicts a fire detection pattern with 2 event sources: Smoke and Temperature. Temperature data is collected by a sensor. The smoke sensor will send an event in case of smoke detection. This data is sent to the system where, within a window of 5 minutes, as soon as the system detects smoke and the temperature exceeds 45, a notification is sent to the alarm. Each new event will therefore trigger a notification. This pattern is inspired by the article [23].

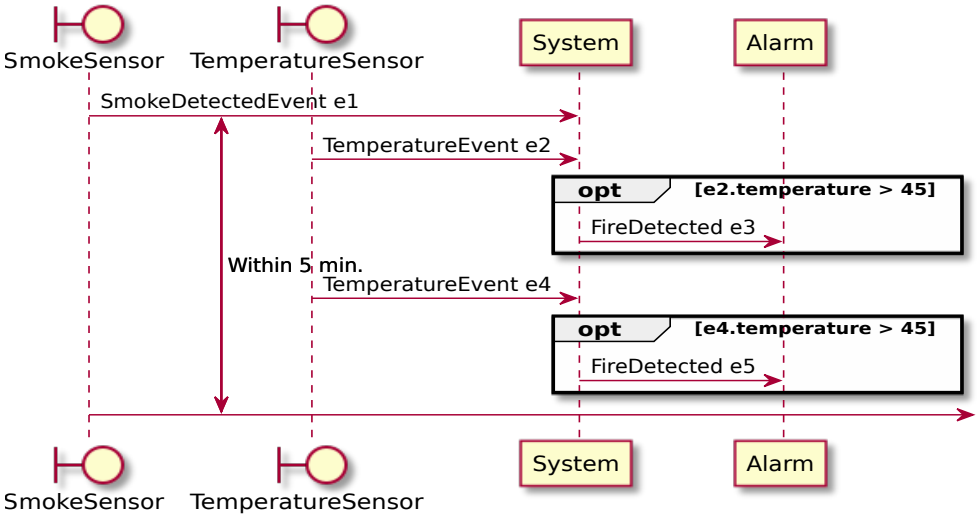

Figure 13: Smart Fire Detection II sequence diagram.

5.3.5 Fire Detection III (Selection, Window, Conjunction, Negation)

Figure 14 illustrates a pattern with 3 event sources: Temperature, Smoke and Rain. Temperature data is collected by a sensor. The smoke sensor will send an event in case of smoke detection. The rain sensor will send events when it rains. This data is sent to the system where, within a 5 minute window, as soon as the system detects smoke, the temperature exceeds 45 and there is no rain, a notification will be sent to the alarm. In addition, each event must be correlated with the room in the house. Each event therefore returns an additional location data. This pattern is inspired by the article [23].

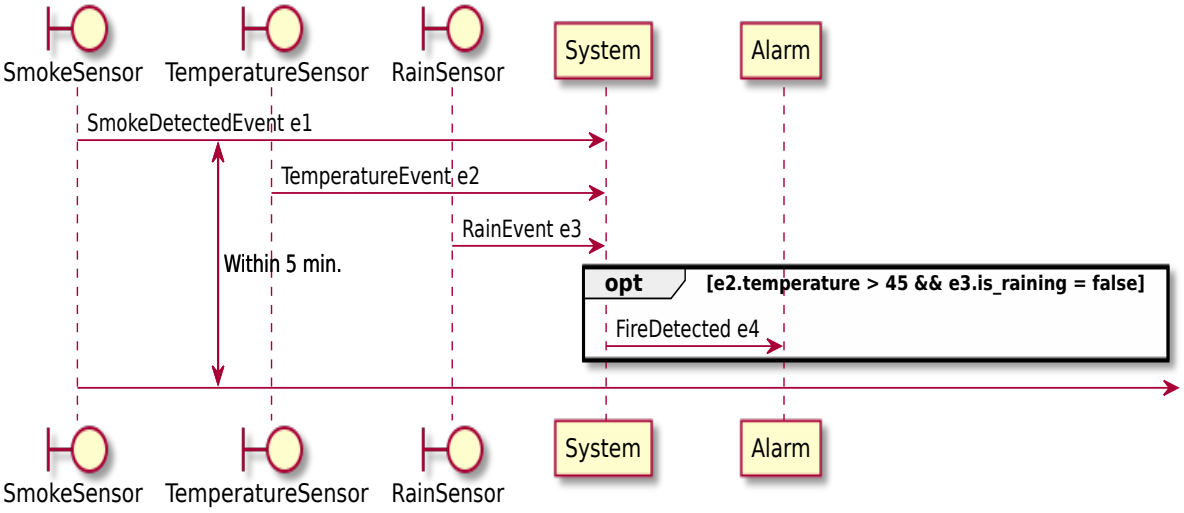

Figure 14: Smart Fire Detection III sequence diagram.

5.3.6 Dangerous Location (Selection, Repetition)

Figure 15 depicts as simple scenario from only one event source: Crash. When the system receives 100 Crash event associated to a location, an event called Dangerous Location is sent. This pattern is inspired by the article [4].

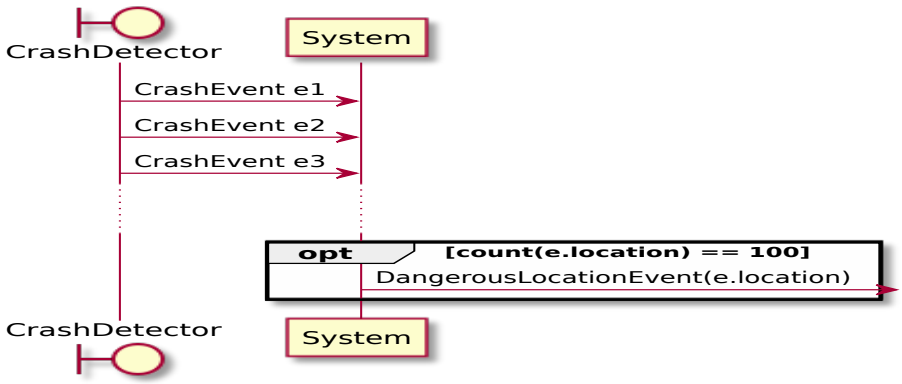

Figure 15: Dangerous Location sequence diagram.

5.3.7 Accident Detection (Selection, Window, Sequence)

Figure 16 illustrates a scenario from 3 event sources: Tire, Crash and Seat. Within 3 seconds, if the system receives an exploded tire event followed by a crash event and there is no longer a presence in the seat, an accident event will occur. This pattern is inspired by article [4].

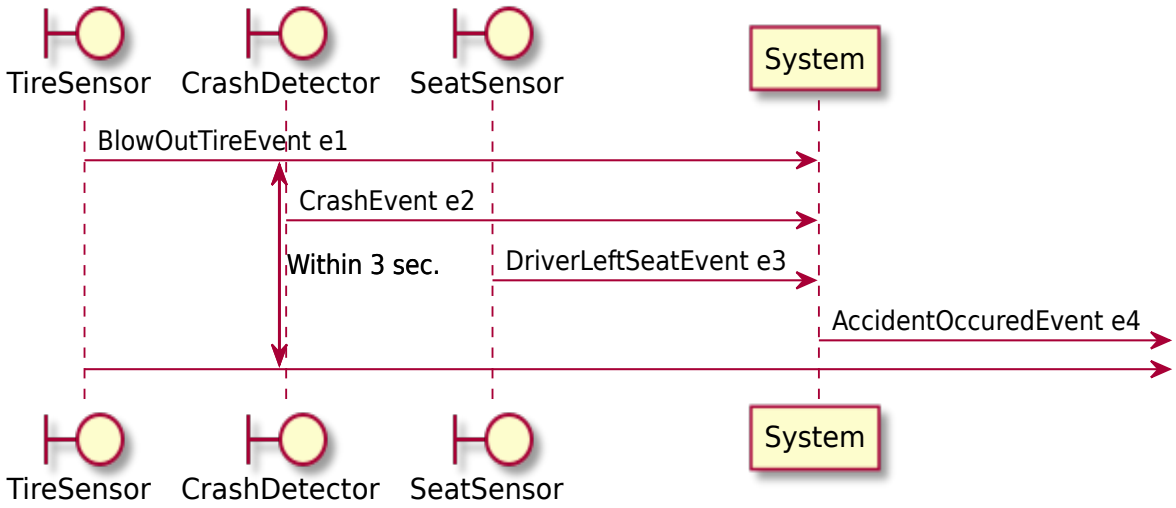

Figure 16: Accident Detection sequence diagram.

6 Results and Implementations

This section has two objectives. The first objective is to fill in the different metrics raised in the section 5.2 for each of the analyzed frameworks. The second objective is to give a rather global feedback on the framework. This feedback will focus primarily on the complexity and/or ease of implementing the CEP patterns of each framework. The details of code implementation will not be analyzed in this paper because this is not the goal of this thesis. On the other hand, they can be found on the GitHub repository `https://github.com/warken2903/-IHDCM050-Complex-Event-Processing-for-IoT` in the directory *Project* for readers who would be interested.

6.1 System Characteristics

All the results to be presented were obtained by running the tests on the following machine:

- **Operating System:** Windows 10 Family 64 bits
- **CPU:** Intel Core i7-6700HQ @ 2.60GHz
- **GPU:** NVidia GeForce 970M 3072Mo GDDR5
- **RAM:** 16.0 Go DDR3

6.2 Esper

6.2.1 Results

General Metrics

This part will highlight some general aspects to take into account when setting up a project with the Esper framework.

Documentation

- Architecture: The documentation presents in a clear way the functioning of the Esper framework. It also presents the context in which this framework is integrated.
- Quick Start: In the official documentation, the first section is "getting started" which allows to setup a first project using Esper.
- CEP Operations: In the documentation provided by Esper, each operator is explained and is illustrated with an example.
- Concrete use-cases: The documentation has two sections dedicated to a set of short or long use-cases. These examples are accompanied by CEP patterns and explanations. There is also a FAQ and solution pattern section that allows you to find questions you might have when writing a CEP pattern.

Support

- Personal support: Esper provides support for two cases: development and production. For development, support answers how-to questions and does not fix software bugs. They help for EPL or application developer get the job done but its purpose is not to provide training and its purpose is not to deliver a fully-developed and tested final solution. It can help only on the current version of Esper. It is contactable via internet and e-mail. As far as production support is concerned, 3 formulas are proposed which cover the last year of versions. They are contactable via web and e-mail. The Silver and Gold package allows you to contact them by phone.
- Community and Forums: There is a way to signal bugs by opening an issue on GitHub. In order to communicate with community, it's possible to ask a question on Stack Overflow.

Portability

- Programming Languages: JVM and .NET compatible languages
- Operating System: For the Java library, Java Virtual Machine compatible operating systems such as Windows, Linux and MacOS are supported. For .NET libraries, Windows is supported. It is also possible to integrate Esper with .NET Core and can therefore run on Linux and MacOS.

Code Metrics

Lines of code

As shown in the table 1, a CEP scenario is less than 5 lines long for medium complexity CEP scenarios. For the Smart Lightning pattern, it has been decomposed into 2 patterns. Lightning detection is done in 2 lines. The detection of lights off is done in 3 lines.

Accident Detection	4 Lines
Dangerous Location	3 Lines
Fire Detection I	4 Lines
Fire Detection II	4 Lines
Fire Detection III	4 Lines
Smart Lightning	3 + 2 Lines
Water Level Overflow	3 Lines

Table 1: Line of codes for Esper CEP Pattern

Technical Metrics

This section aims to measure the technical metrics of the Esper framework. CPU, RAM and disk metrics have been measured. Regarding the GPU, this framework does not use the graphics card to process events.

CPU Usage:

Regarding the use of the processor, it remains stable despite the intensity of the received event. The average usage is below 22%. We can notice on the figure 17 that there is an increase in processor usage for the Fire Detection III pattern. The other patterns tend towards the same consumption.

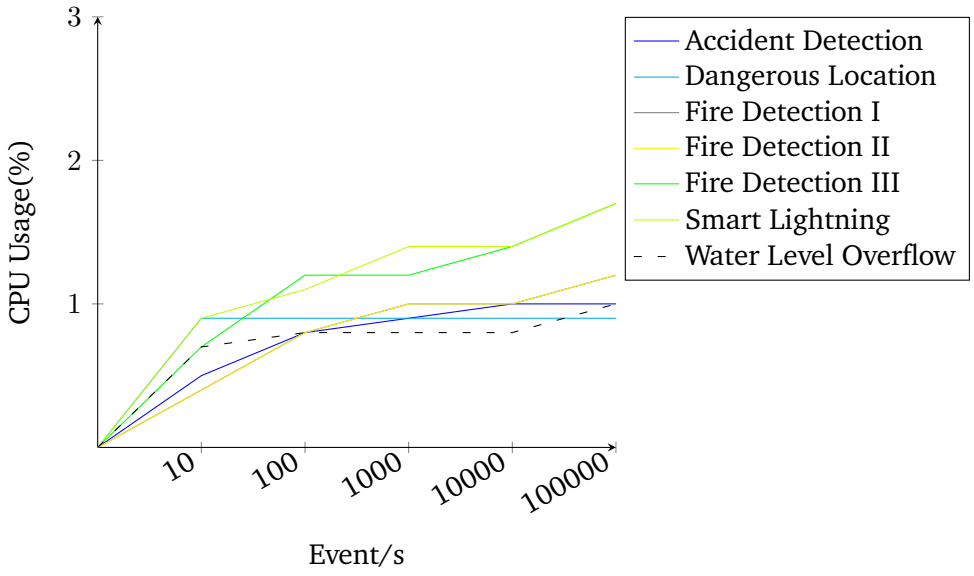

Figure 17: Esper CPU Usage

RAM Usage:

Regarding the use of RAM, Esper does not show any significant increase in the intensity of events received. The figure 18 illustrates this well and shows that the RAM memory used is stagnating at 1.5GB.

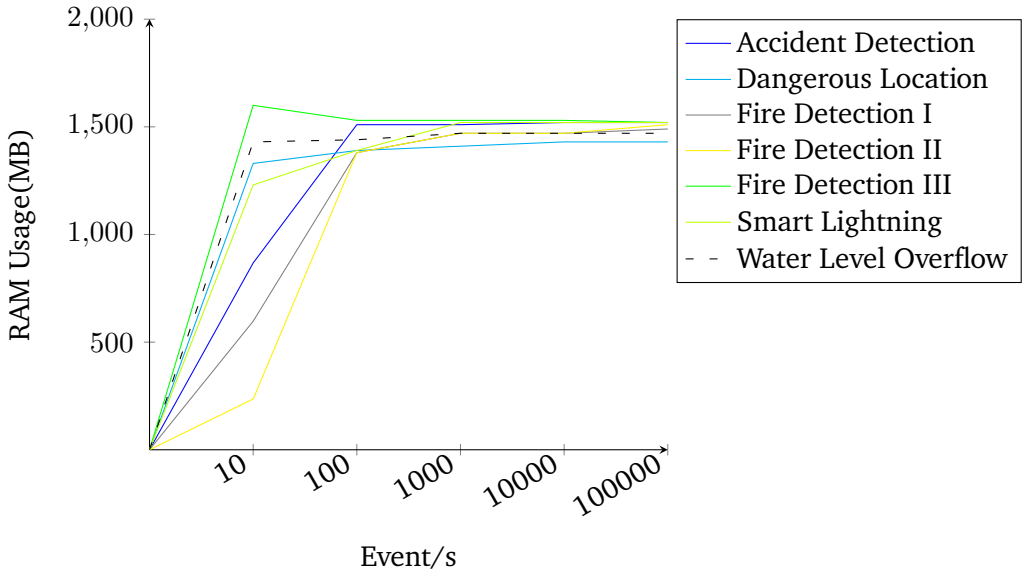

Figure 18: Esper RAM Usage

Disk Usage:

Regarding disk usage, figure 19 illustrates that disk usage increases with the rate at which events are sent. For most patterns, this usage tends to be 10MB. However, for the Smart Lightning pattern, this usage tends to 25MB.

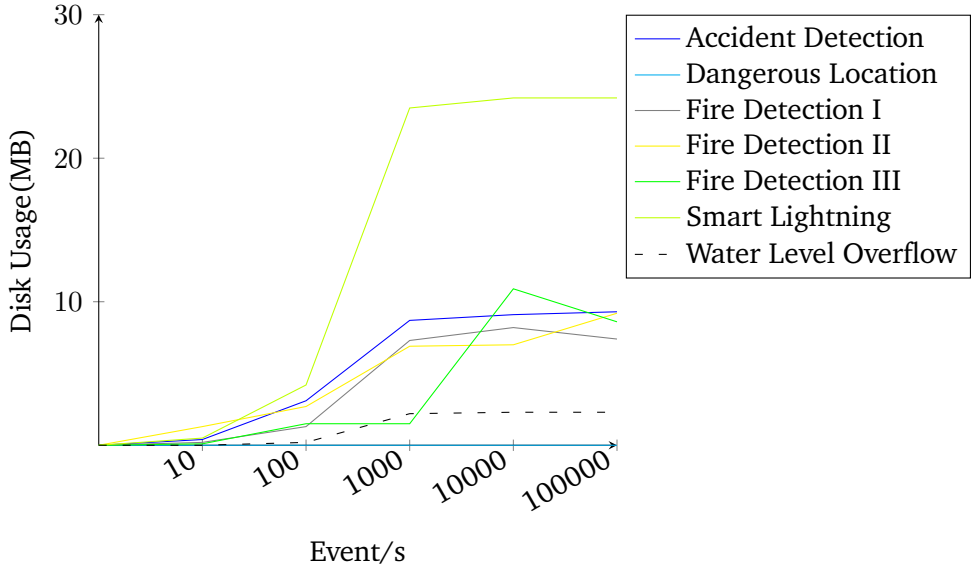

Figure 19: Esper Disk Usage

6.2.2 Implementation

Esper has a very complete documentation. There are really a lot of very concrete examples with implementation details. It's quite easy for a developer to understand and use it to adapt it to his business rules. There's a FAQ section that answers a lot of questions you may have during implementation. However, the repository also has examples but these are not very clear. The maven package structure is very particular and it is difficult to make them work. (link: `https://github.com/espertechinc/esper/tree/master/examples`)

Another problem is that there are 8 major versions of Esper. This implies that problems raised by the community, on Stack Overflow for example, are not necessarily latest. It is therefore difficult to find an answer to a question in the latest version. Regarding development time, 15 hours had to be invested in setting up the project and implementing the different CEP patterns. For developers who have a knowledge of the EPLs, there will be no difficulty to implement them in Esper.

Overall, Esper facilitates software development that requires Complex Event Processing techniques. Despite the 8 major versions of this framework, this one has a rather fast handling thanks to the numerous examples. The documentation is also updated each time compared to the current version. The documentation is also versioned and it is therefore possible to find the documentation corresponding to the version used.

6.3 Siddhi

6.3.1 Results

General Metrics

This part will highlight some general aspects to take into account when setting up a project with the Siddhi framework.

Documentation

- Architecture: On Siddhi's official website, the high level architecture is explained. Therefore, it is easy to see where Siddhi can fit into an existing environment.
- Quick Start: Siddhi's official website has a section dedicated to a quick start guide that explains how to use Siddhi for the first time. In other sections, the different ways to use Siddhi are also explained in detail. (such as a library, Docker, etc.)
- CEP Operations: The documentation provides an exhaustive list of all operators available with Siddhi with examples. All the concepts needed to use this framework are also explained.
- Concrete use-cases: Siddhi's official website exposes some specific use-cases. Each of these use-cases is explained with a step-by-step methodology to meet the need initially stated.

Support

- Personal support: There is no personal support provided by Siddhi directly. On the other hand, the company WSO2, which is Siddhi's main contributor, offers personal support.
- Community and Forums: All communication with Siddhi is done through different public tools. There are two Google groups, one for developers/contributors and the other is dedicated to questions from Siddhi users. The community is also available via Slack. For any question, it is possible to call upon the community via GitHub issues as well as via Stack Overflow.

Portability

- Programming Languages: Java and Python as library
- Operating System: Windows, Linux and MacOS. Siddhi applications are easily containerizable and can therefore run on OSes that support containerization.

Code Metrics

Lines of code
As shown in the table 2, a CEP scenario is a maximum of 5 lines for CEP scenarios of medium complexity. For the Smart Lightning pattern, it has been decomposed into 2 patterns. Lightning detection is done in 2 lines. The detection of lights off is done in 3 lines.

Accident Detection	3 Lines
Dangerous Location	4 Lines
Fire Detection I	4 Lines
Fire Detection II	5 Lines
Fire Detection III	5 Lines
Smart Lightning	3 + 2 Lines
Water Level Overflow	3 Lines

Table 2: Line of codes for Siddhi CEP Pattern

Technical Metrics

This section aims to measure the technical metrics of the Siddhi framework. CPU, RAM and disk metrics have been measured. Regarding the GPU, this framework does not use the graphics card to process events.

CPU Usage:

Concerning the Siddhi framework, the figure 20 indicates that CPU consumption remains mostly stable (below 1%). However, we can notice that for the Fire Detection III pattern, the CPU consumption increases up to 2.5% from 1000 events per second.

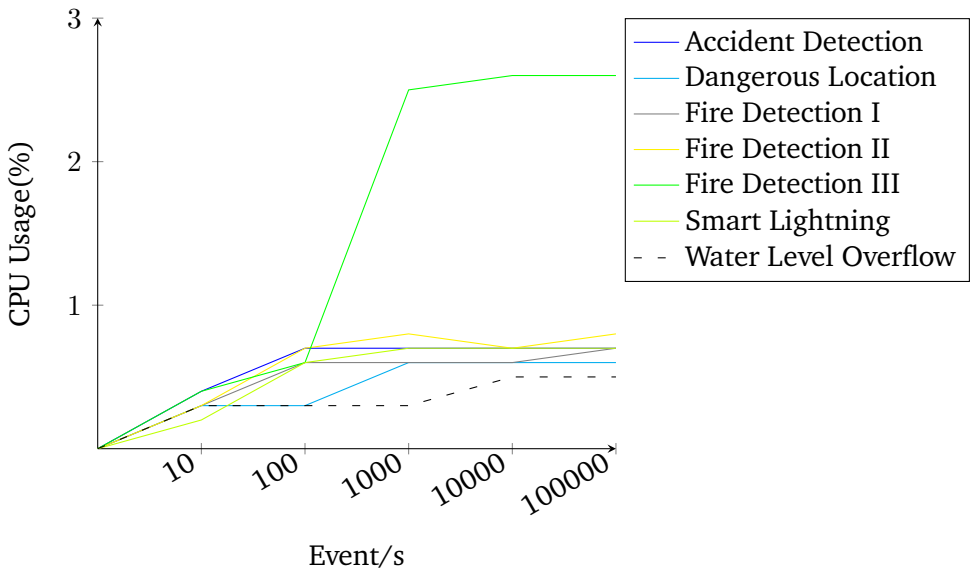

Figure 20: Siddhi CPU Usage

RAM Usage:

RAM usage varies between 500MB and 1.1GB. (see figure 21) There is no constant growth as a function of event sending intensity. RAM usage remains stable.

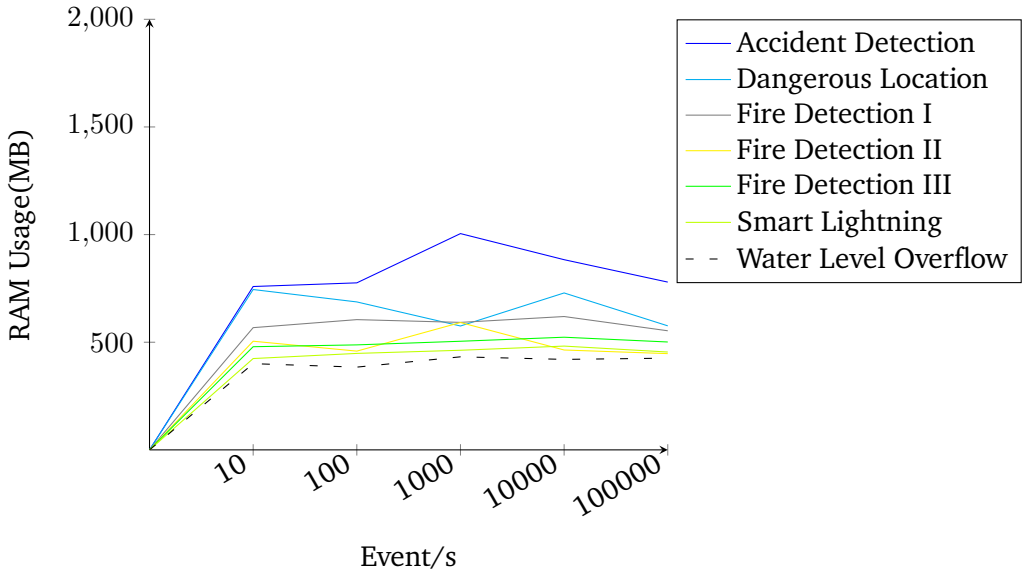

Figure 21: Siddhi RAM Usage

Disk Usage:

In terms of disk usage, there is a slight increase depending on the patterns used. As shown in figure 22, this consumption remains below 10 MB.

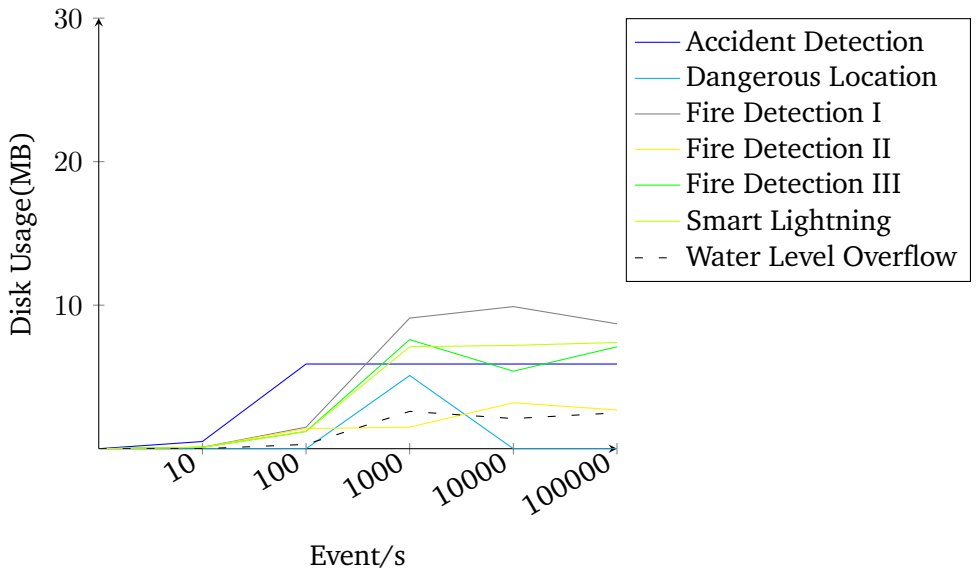

Figure 22: Siddhi Disk Usage

6.3.2 Implementation

Siddhi's Documentation is complete and explains in detail the available operators with an associated example. Siddhi offers a lot of tools including a graphical tool to facilitate the writing of EPLs. The DSL proposed by Siddhi is really powerful because only from a .siddhi definition, which actually consists in the definition of patterns with also the definition of output stream, an application can be created. There is a tool that allows to launch this file as a "microservice", you can link it to a database, define the protocol and format of the output streams. The different deployment modes are explained very clearly in the documentation and videos for this purpose are available.

Overall, Siddhi is very intuitive to use and offers easy integration with third party services by being agnostic in the implementation of the .siddhi file. This makes it easier to change the underlying infrastructure without changing the code. However, there is no template for the embedded library. This framework is mainly focused on running the .siddhi file as a "microservice". Their eco-system provides what is needed in the development of a CEP project. Regarding development time, 8 hours had to be invested in setting up the project and implementing the different CEP patterns.

6.4 Drools

6.4.1 Results

General Metrics

This part will highlight some general aspects to take into account when setting up a project with the Drools framework.

Documentation

- Architecture: The Drools documentation presents the objectives of the framework. However, it is very complicated to find your way around in the documentation. There are many sub-projects and it is sometimes complicated to know where to look.

- Quick Start: There is no official Quick Start Guide for Drools Fusion.
- CEP Operations: Drools Fusion CEP operations are described in the documentation with corresponding examples. All important concepts for Complex Event Processing in Drools are also explained.
- Concrete use-cases: There is no concrete use-cases. On the GitHub repository, there's only some code examples but they are complex to make them work on local machine. The way to compile them is not standard and the scripts given weren't working during tests.

Support

- Personal support: There is paid personal support available. For this service, it is possible to contact them via the web or via phone. They offer 2 different formulas, one of which proposes contact by phone. (https://www.drools.org/product/services.html)
- Community and Forums: To ask a question to the community, it is possible to go through the Stack Overflow platform. If you want to report a bug, you can create an issue in Jira. For contribution to the project, 2 Google Groups are available. To follow the development discussions, a Zulip chat is also available.

Portability

- Programming Languages: Java
- Operating System: Windows, Linux, MacOS

Code Metrics

Lines of code
As shown in the table 3, a CEP scenario is less than 5 lines for medium complexity CEP scenarios. For the Smart Lightning pattern, it has been decomposed into 2 patterns. Lightning detection is done in 1 line. The detection of lights off is done in 2 lines.

Accident Detection	3 Lines
Dangerous Location	4 Lines
Fire Detection I	4 Lines
Fire Detection II	3 Lines
Fire Detection III	4 Lines
Smart Lightning	2 + 1 Lines
Water Level Overflow	1 Line

Table 3: Line of codes for Drools CEP Pattern

Technical Metrics

This section aims to measure the technical metrics of the Drools framework. CPU, RAM and disk metrics have been measured. Regarding the GPU, this framework does not use the graphics card to process events.

CPU Usage:

Concerning the Drools framework, its CPU usage increases significantly depending on the number of events received. The figure 23 shows that there is also a difference in CPU usage depending on the pattern. For 100,000 events per second, the average CPU consumption rises to 13%. However, the application saturates at 1000 events per second for Fire Detection II and III patterns.

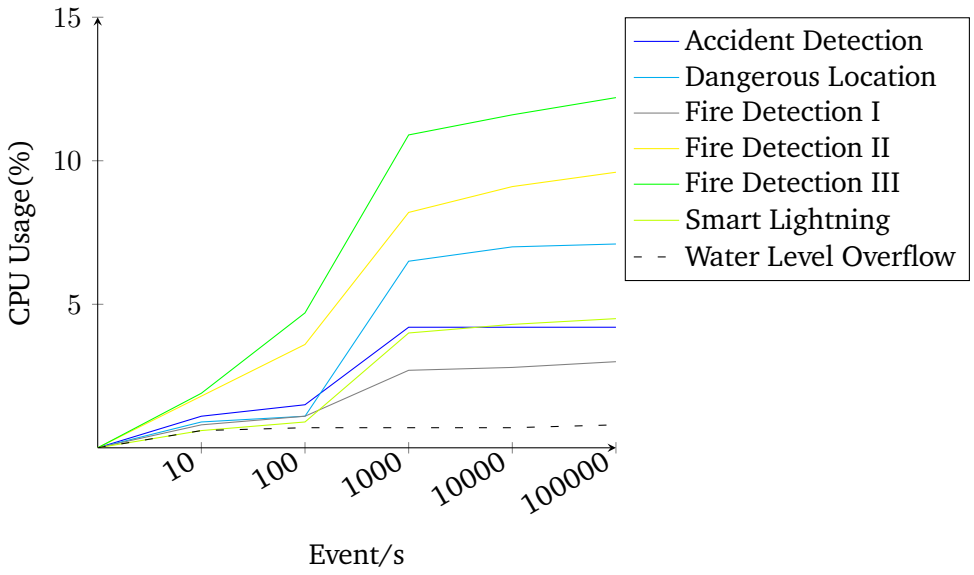

Figure 23: Drools CPU Usage

RAM Usage:

Concerning the Drools framework, its use in RAM remains mostly stable at 1GB. However, as shown on the figure 24, Fire Detection II and III patterns show a very large increase from 1000 events per second. (> 3.0GB) This makes the application unusable.

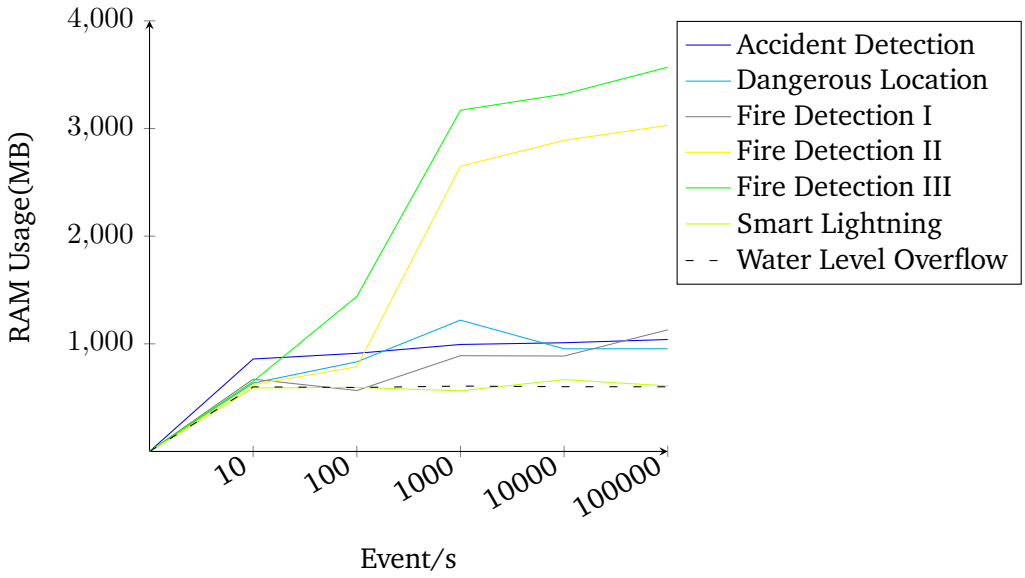

Figure 24: Drools RAM Usage

Disk Usage:

Regarding disk usage, it remains stable below 10MB for most patterns. For Fire Detection II and III patterns, there is a very large increase to 100 events per second (cf. 25) After that, the application is unusable for these patterns. This is why the disk usage decreases.

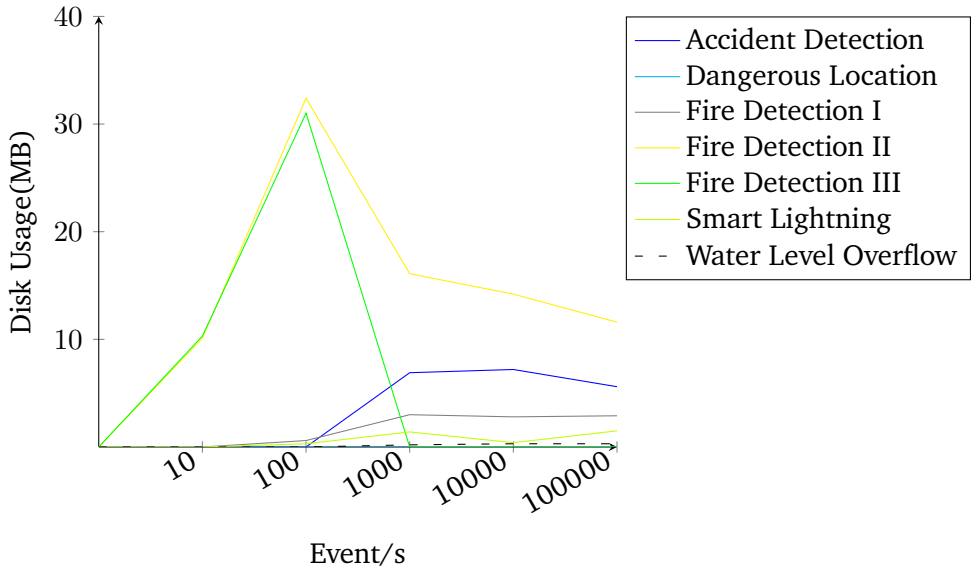

Figure 25: Drools Disk Usage

6.4.2 Implementation

Drools offers an approach quite different from other Complex Event Processing frameworks. Moreover, it is a rule engine that offers Complex Event Processing functionalities. The detection system is not designed like the one of Esper or Siddhi. The EPL language used by Drools is a mixture of Java and semantics added by Drools. It does not look like a SQL-like language as seen for Esper or Siddhi. This gives more flexibility to Drools because it is possible to use Java classes if you need to do particular operations that would be complicated to do in SQL.

The documentation of Drools is very extensive because Drools offers a lot of external tools and libraries. It is sometimes difficult to find the information you are looking for in the documentation. During a first implementation, it is also quite complicated to find out how to run the library because there is no quick start guide. However, there are many public repositories where the community has published examples. It took 18 hours to set up the project and implement the CEP patterns.

In addition, this framework poses certain problems. One problem with this framework is that it will keep in memory the events and then it will reapply the rules on this stream. It will therefore for each execution, re-interpret the rules and thus potentially return events already passed. This problem occurs when we use a sliding time window. There is no intermediate state that Drools will keep in order not to re-apply rules that have already been sent. The Drools framework does not seem really adapted to apply the rules to each new event received. If we want to be able to do this, we must think about setting up a system of idempotence at the level of the action to send and / or at the application level. (e.g. with a unique identifier)

6.5 Perseo

6.5.1 Results

General Metrics

This part will highlight some general aspects to take into account when setting up a project with the Drools framework.

Documentation

- Architecture: The documentation explains clearly the big picture of Perseo. It shows the interactions between the different components.
- Quick Start: In the Perseo documentation, there is a section dedicated to deployment. The guide is well explained but is unfortunately not functional. For deployment via Docker, some parameters are no longer updated and therefore the given configuration is not taken into account. There is also no clear guide that explains how to configure the context broker with rules, events, etc.
- CEP Operations: The Complex Event Processing engine used is Esper. The official Esper documentation explains very clearly the different operators. However, Perseo has some peculiarities that should be better highlighted. It is not always clear what is possible or not to do with this framework.
- Concrete use-cases: There are no concrete use cases.

Support

- Personal support: There is no personal support.
- Community and Forums: It's possible to raise a question on Stack Overflow or open an issue on GitHub if a bug is detected.

Portability

- Programming Languages: There is no embedded library. Perseo rules are based on Esper.
- Operating System: Linux, Windows, MacOs and any OS supporting containerization

Code Metrics and Technical Metrics

It is not possible to measure these 2 categories of metrics for this framework. The rules based on Esper are not interpreted correctly. (cf. section 6.5.2)

6.5.2 Implementation

The environment is very tedious to set up. Some places in the documentation are not up to date. This leads to confusion when setting up the containerised environment because some parameters have been renamed and the old ones are not taken into account. This leads to a lot of questions and searching in forums for only the bootstrapping of services. There are a lot of services and therefore dependencies, the architecture should be better explained in the documentation.

The rules are based on Esper. So the formalism is very similar, the only downside is that you have to cast the types to be able to compare the data and the rule can be valid. The semantics are really complicated to understand, the initial project has no good documentation. It is confusing,

the initial project is maintained but documentation is provided by a third party who forked the project.

Despite this, all the services in the stack provided in the documentation could be deployed correctly after many manipulations and parameter modifications. However, when first implementing the *Water Level Overflow* pattern, which is a very simple pattern. The realized pattern is validated by the application but it is never triggered when sending an event. The ">" operator does not seem to work. When an event with a water level greater than the guard is sent, the rule is not executed. By removing this condition, the rule is triggered. This prevented the implementation of the other CEP patterns. In order to get to this stage, it took 20 hours to reach a result that is not conclusive. Different versions have been tested but none of them were conclusive.

7 Interpretation and Discussion of Results

The objective of this section is to compare the different frameworks analysed on the basis of the 3 axes defined in the criteria. With the results obtained, it will be possible to answer the problematic posed in the context of this thesis.

7.1 General Metrics

Concerning the documentation, on the table 4 appears that the Esper and Siddhi framework offer a very high quality of documentation. As for Drools, it has a good documentation on the general architecture and on the CEP operators. On the other hand, the handling is more complex because it does not provide a Quick Start guide and concrete use cases. Perseo has a good explanation of the architecture and has a Quick Start guide. Unfortunately, this is not updated and therefore leads to errors on the part of the developer. The documentation on the Perseo EPL language is also not very comprehensive. Although this framework uses Esper as EPL language, they add a part of generic that is not well documented. Perseo also does not provide a concrete use case.

Concerning the support, the table 4 shows that Esper, Siddhi and Drools offer practically the same quality of support. The difference is in the communication with the community. Esper has only 2 channels of communication with the community while Siddhi and Drools offer more than 3. On the other hand, Perseo does not offer personal support and like Esper, Perseo offers only 2 means of communication with the community.

Concerning portability, the table 4 indicates that all frameworks are supported by many operating systems. Regarding programming languages, Esper and Siddhi are compatible with 2 programming languages while Drools offers only one. For Perseo, it's quite different because there is no embedded library.

In summary, Esper and Siddhi are the 2 frameworks that offer a very good documentation and thus a rather simple and fast handling. Both frameworks offer good support. Siddhi offers slightly more communication with the community than Esper. Finally, both Esper and Siddhi offer the same level of quality regarding portability.

	Esper	Siddhi	Drools	Perseo
Documentation				
Architecture				
Quick Start				**Not fully functional**
CEP Operations				**Not exhaustive**
Concrete Use-Cases				
Support				
Personal Support		**WSO2 support**		
Community and Forums	**2**	**>3**	**>3**	**2**
Portability				
Programming Languages	**2**	**2**	**1**	**No embedded Library**
Operating System	**>=3**	**>=3**	**>=3**	**>=3**

Table 4: General Metrics Summary

7.2 Code Metrics

Concerning the lines of code, the table 5 shows very clearly that there are no drastic differences between frameworks. For the most part, Drools takes one or two lines of code less than its competitors. However, this difference is very small and is not very useful in the choice between these 3 frameworks.

In summary, Esper, Siddhi and Drools offer the same line load whatever the pattern.

	Esper	**Siddhi**	**Drools**
Accident Detection	4	3	3
Dangerous Location	3	4	4
Fire Detection I	4	4	4
Fire Detection II	4	5	3
Fire Detection III	4	5	4
Smart Lightning	5	5	3
Water Level Overflow	3	3	1

Table 5: Code Metrics Summary

7.3 Technical Metrics

In order to have a better vision on the technical metrics, it is interesting to make several summary tables per resource with a summary of the results obtained by framework.

Regarding CPU consumption, we notice on the table 6 that Drools has a higher consumption than the other two frameworks. It should also be taken into account that when a high load is reached, Drools is no longer usable. For Fire Detection II and III patterns, Drools doesn't respond anymore and consumes a lot of resources. Esper and Siddhi have a relatively equivalent consumption. Siddhi tends to consume slightly less CPU except for the Fire Detection III pattern.

In summary, Esper and Siddhi have a CPU consumption that is low and remains stable even under load. Drools does not prevent itself from a big load increase.

	Esper	Siddhi	Drools
Accident Detection	1.0 %	0.7 %	4.2 %
Dangerous Location	0.9 %	0.3 %	7.1 %
Fire Detection I	1.2 %	0.7 %	3.0 %
Fire Detection II	1.2 %	0.8 %	9.6 %
Fire Detection III	1.7 %	2.6 %	12.2 %
Smart Lightning	1.7 %	0.7 %	4.5 %
Water Level Overflow	1.0 %	0.5 %	0.8 %

Table 6: Maximum CPU Usage Summary

Concerning the RAM consumption, we notice on the table 7 that Drools has the highest peak RAM usage. However, apart from Fire Detection II and III patterns, Drools has a lower RAM consumption than Esper. Regarding the Siddhi framework, the maximum RAM used is 1005 MB, which is 500 MB less than Esper.

In summary, the framework consuming the least amount of RAM is Siddhi. Esper and Drools consume a bit more RAM but their consumption is still acceptable.

	Esper	Siddhi	Drools
Accident Detection	1520 MB	1005 MB	1040 MB
Dangerous Location	1430 MB	744.8 MB	1220 MB
Fire Detection I	1490 MB	619.4 MB	1130 MB
Fire Detection II	1510 MB	591.5 MB	3030 MB
Fire Detection III	1520 MB	523.3 MB	3570 MB
Smart Lightning	1520 MB	482.1 MB	667.6 MB
Water Level Overflow	1470 MB	426 MB	607.1 MB

Table 7: Maximum RAM Usage Summary

Concerning disk consumption, we notice on the table 8 that most of the consumption is below 10 MB. However, for the Drools framework, the Fire Detection II pattern had a consumption of 2820 MB. As already mentioned, Fire Detection II and III patterns are unstable with the Drools framework when the received events load is high.

Globally, Siddhi uses the least amount of disk resources in general. Esper has a peak of 24.2MB but the load is well supported with heavy workloads.

	Esper	Siddhi	Drools
Accident Detection	9.3 MB	5.9 MB	7.2 MB
Dangerous Location	0.0 MB	5.1 MB	0.0 MB
Fire Detection I	8.2 MB	9.9 MB	3.0 MB
Fire Detection II	9.2 MB	3.2 MB	2820 MB
Fire Detection III	8.6 MB	7.6 MB	31.0 MB
Smart Lightning	24.2 MB	7.4 MB	1.5 MB
Water Level Overflow	2.3 MB	2.6 MB	0.3 MB

Table 8: Maximum Disk Usage Summary

8 Conclusion

The purpose of this thesis was to compare different frameworks of Complex Event Processing Open-Source still maintained today, based on a grid of metrics that has 3 axes: getting started, maintainability and resource usage.

First, we had to find a generic way to compare frameworks between them. It turns out that the solution that was chosen was to define scenarios that each framework should be able to respond to. Then, it was necessary to define metrics for each of the 3 axes mentioned above. In order to be able to capture metrics regarding resource consumption, it was decided to define test scenarios with load scaling. Once this part of the definition was completed, it was possible to realize the implementations of the scenarios in the selected frameworks as well as to realize the load tests. Once the tests were completed, the general metrics as well as the code metrics were captured.

With all these results, it was possible to highlight the strengths and weaknesses of each framework according to these 3 axes. The information that emerged from the analysis of the results will help to answer the various problem questions stated at the beginning of this paper:

Which framework allows to develop an IoT solution in the easiest way ? The Esper and Siddhi frameworks both address this problematic.

Which framework allows to develop the most maintainable IoT solution ? On this aspect, Esper, Siddhi and Drools cannot be distinguished. Each of these frameworks offers the same quality of maintainability.

Which framework allows to develop an IoT solution that uses the least amount of resources ? The Siddhi framework unanimously uses the least amount of CPU, RAM and Disk resources. Note that the Esper framework uses slightly more CPU and RAM than Siddhi but still remains powerful and stable.

Among the improvements that could be made in this research is the addition of a 4th axis concerning the performance of frameworks that could be very interesting. Indeed, it is an important criterion in the scope of a project. This 4th dimension could refine the results of the research. In addition, an improvement in the metrics regarding the maintainability aspect of the code could have been achieved. The number of lines of code is not necessarily the only factor in code maintainability complexity.

In future research, the analysis of proprietary Complex Event Processing frameworks with the same analysis grid as the one used in this research could be interesting to conduct. Indeed, these two works could be complementary and guide users who want to start a project involving Complex Event Processing. In addition, further work on Event Stream Processing platforms could be carried out. It would be interesting to see how ESP frameworks with Complex Event Processing mechanisms can respond to the patterns proposed in this thesis.

List of Figures

References

[1] M. Bakhouya et al. “Towards a context-driven platform using IoT and big data technologies for energy efficient buildings”. In: *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*. Oct. 2017, pp. 1–5. DOI: 10.1109/CloudTech.2017.8284744.

[2] S. R. Bhandari and N. W. Bergmann. “An internet-of-things system architecture based on services and events”. In: *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. Apr. 2013, pp. 339–344. DOI: 10.1109/ISSNIP.2013.6529813.

[3] Roman Bukarev. *IoT Mashup of Sensors, Cloud Software, and Machine Learning on the MapR Data Platform*. 2018. URL: https://mapr.com/blog/cool-fitness-tracker-using-iot-sensors-cloud-ml-on-mapr (visited on 08/02/2019).

[4] L. Burgueño, J. Boubeta-Puig, and A. Vallecillo. “Formalizing Complex Event Processing Systems in Maude”. In: *IEEE Access* 6 (2018), pp. 23222–23241.

[5] C. Y. Chen et al. “Complex event processing for the Internet of Things and its applications”. In: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. Aug. 2014, pp. 1144–1149. DOI: 10.1109/CoASE.2014.6899470.

[6] *Complex Event Processing : La résurrection ?* URL: https://www.sentelis.com/articles/complex-event-la-r%C3%A9surrection-11-4 (visited on 07/12/2020).

[7] Gianpaolo Cugola and Alessandro Margara. “TESLA: A formally defined event specification language”. In: Jan. 2010, pp. 50–61. DOI: 10.1145/1827418.1827427.

[8] *Decision*. URL: https://github.com/Stratio/Decision/blob/master/README.md (visited on 07/02/2020).

[9] R. Deng et al. “Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing”. In: *2015 IEEE International Conference on Communications (ICC)*. June 2015, pp. 3909–3914. DOI: 10.1109/ICC.2015.7248934.

[10] *Drools*. URL: https://github.com/kiegroup/drools/blob/master/README.md (visited on 07/02/2020).

[11] *Esper*. URL: http://www.espertech.com/esper/ (visited on 07/02/2020).

[12] *FlinkCEP - Complex event processing for Flink*. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/libs/cep.html (visited on 07/02/2020).

[13] D. Giusto et al. *The Internet of Things*. Springer, 2010.

[14] Manufacturers Automation Inc. *Should You Consider Fog Computing for Your IIoT? Moxa*. 2017. URL: https://www.manuauto.com/category/moxa/page/4 (visited on 08/02/2019).

[15] A. A. Ismail, H. S. Hamza, and A. M. Kotb. “Performance Evaluation of Open Source IoT Platforms”. In: *2018 IEEE Global Conference on Internet of Things (GCIoT)*. Dec. 2018, pp. 1–5. DOI: 10.1109/GCIoT.2018.8620130.

[16] Rafiullah Khan et al. “Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges”. In: Dec. 2012, pp. 257–260. ISBN: 978-1-4673-4946-8. DOI: 10.1109/FIT.2012.53.

[17] F. A. Kraemer et al. “Fog Computing in Healthcare–A Review and Discussion”. In: *IEEE Access* 5 (2017), pp. 9206–9222. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2704100.

[18] An Lam and Øystein Haugen. “Complex Event Processing in ThingML”. In: vol. 9959. Oct. 2016, pp. 20–35. ISBN: 978-3-319-46612-5. DOI: 10.1007/978-3-319-46613-2_2.

[19] Edward A. Lee. “Cyber Physical Systems: Design Challenges”. In: (Jan. 2008).

[20] *Logisland*. URL: https://github.com/Hurence/logisland (visited on 07/12/2020).

[21] David Luckham. *What’s the Difference Between ESP and CEP?* 2019. URL: https://complexevents.com/2019/07/15/whats-the-difference-between-esp-and-cep-2/ (visited on 07/02/2020).

[22] David Luckham and Roy W Schulte. “Event Processing Glossary - Version 2.0”. In: (2011).

[23] Kaya Mahir and Çetin-Kaya Yasemin. “Complex Event Processing Using IoT Devices Based On Arduino”. In: Dec. 2017, pp. 13–24. DOI: 10.5121/ijccsa.2017.7602..

[24] D. Mijić and E. Varga. “Unified IoT Platform Architecture Platforms as Major IoT Building Blocks”. In: *2018 International Conference on Computing and Network Communications (CoCoNet)*. Aug. 2018, pp. 6–13. DOI: 10.1109/CoCoNet.2018.8476881.

[25] *Perseo context aware cep*. URL: https://perseo.readthedocs.io/en/latest/#perseo-context-aware-cep (visited on 07/02/2020).

[26] Kai Petersen et al. “Systematic Mapping Studies in Software Engineering”. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering* 17 (June 2008).

[27] D. Robins. “Second International Workshop on Education Technology and Computer Science”. In: 2010.

[28] L. Roffia et al. “A Semantic Publish-Subscribe Architecture for the Internet of Things”. In: *IEEE Internet of Things Journal* 3.6 (Dec. 2016), pp. 1274–1296. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2587380.

[29] I. Schmerken. *Deciphering the myths around complex event processing*. Wall Street & Technology, May 2008.

[30] W. Shi et al. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016), pp. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[31] *Siddhi Core Libraries*. URL: https://github.com/siddhi-io/siddhi/blob/master/README.md (visited on 07/02/2020).

[32] *TSP*. URL: https://github.com/Clover-Group/tsp (visited on 07/12/2020).

[33] Paul Vincent. *CEP: more than event patterns*. 2009. URL: https://www.tibco.com/blog/2009/12/18/cep-more-than-event-patterns (visited on 07/28/2019).

[34] *What is Apache Flink? — Applications*. URL: https://flink.apache.org/flink-applications.html (visited on 07/02/2020).

[35] *wso2 Complex Event Processor*. URL: https://github.com/wso2-attic/product-cep/blob/master/README.md (visited on 07/02/2020).

[36] Wu Y et al. “RFID enabled traceability networks: A survey”. In: (2011).